

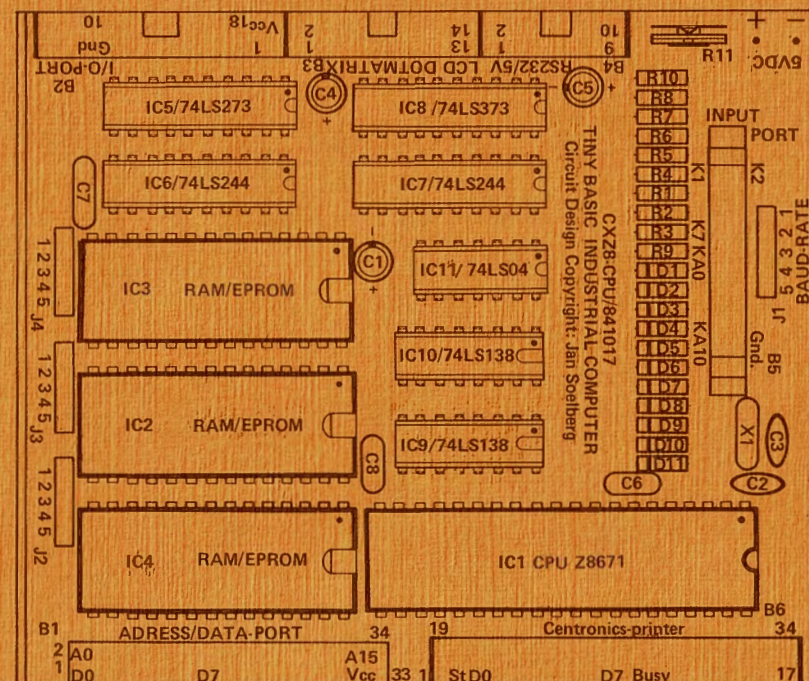
Z8 COMPUTER DESIGN

CXZ8-CPU/841120 - Z8-COMPUTER FOR STYRING & REGULERING

Copyright: Jan Soelberg - Circuit Design - Danmark

ISBN: 87-7383-017-8 / Karlstrupgaard november 1984 / 1. udgave

DANSK ELEKTRONIK



Z8 COMPUTER DESIGN

Af: JAN SOELBERG



CXZ8-CPU

Z8 TINY BASIC FOR STYRING & REGULERING

CXZ8-CPU

Bestillingsnummer for udgivelse:
CM50-LFV - ISBN: 87-7383-017-8

Circuit Design Forlag: Den 20-11-1984
Ide, design, sats, layout, montage, produktion og salg: Circuit Design
Tryk & repro: H.C. Jensen Offset - Karlslunde
Bogbinding: Jean Christensen - Karlslunde

Udgivet i forbindelse med lanceringen af Z8 TINY-BASIC -computeren til styring, regulering eller specielle EDB-opgaver. Denne beskrivelse danner grundlaget for videre udbygninger og software, som introduceres i medlemsbladet CIRCUIT i 1985. Oplysninger om hard- og software kan samles i specialringbindet THE CXZ8 MANUAL, som fås fra forlaget (varenummer: LZ8-M). Denne publikation er udstyret med specialringe i rykken, som passer i det omtalte ringbind. CD forbeholder sig COPYRIGHT på CXZ8-CPU printpladen, men IKKE på funktion og IKKE på software. Derfor kan computeren frit anvendes i kommerciel produktion og frit dokumenteres på grundlag af CD's oplæg. Eneste restriktion er at Circuit Design i hvert tilfælde skriftligt giver tilladelse til kommerciel udnyttelse (gratis) og at Circuit Design får stillet et eksemplar af den udviklede brugersoftware til rådighed. Denne software vil IKKE blive stillet til rådighed for andre uden aftale.

Karlstrupgaard den 20-11-1984

TINY BASIC MICROCOMPUTER

EN LILLE DEDIKERET COMPUTER

1.1	FORMÅL	4
1.2	Tekniske muligheder	5
1.3	Udvidelsesmuligheder	5

DET PRAKTISKE ARBEJDE

2.0	Opbygning	6
2.1	Memory organisering	6
2.2	Diagram	9(28)
2.3	Perifere tilslutninger	8
2.4.0	RS232/5V kommunikation	9
2.4.1	VIC20 program	12
2.4.2	CBM64 program	13
2.4.3	Spectrum 48K program	16
2.5	Komponentliste & samling af CPU	19

BASIC/DEBUG

3.1.1	Design forudsætninger	22
3.1.2	Basic/Debug afviklings måder	22
3.1.3	Programlinje syntax	23
3.1.4	Basic/Debug editor	23

BASIC/DEBUG UDTRYK

3.2.1	Introduktion	24
3.2.2	Talbehandling	24
3.2.3	Konstanter	25
3.2.4	Variabler	25
3.2.5	Operationer	25
3.2.6	Hukommelse/Memory/Adresser	26
3.2.7	Funktioner	27
3.2.7.1	Logiske funktioner	27
3.2.7.2	Maskinkode funktioner	30
3.2.8	Syntax ved operationer	30

BASIC/DEBUG UDTRYK & DEFINITIONER

3.3.1	Introduktion	31
3.3.2	GO@ kommando	31
3.3.3	GOSUB kommando	32
3.3.4	GOTO kommando	32
3.3.5	IF/THEN kommando	33
3.3.6	INPUT og IN kommandoer	34
3.3.7	LET kommando	35
3.3.8	LIST kommando	36
3.3.9	NEW kommando	37
3.3.10	PRINT kommando	37
3.3.11	PRINT kommando	39
3.3.12	REM kommando	39
3.3.13	RUN kommando	39
3.3.14	STOP kommando	40

```

6240 01 EB 3C E6 24 00 E6 25
6248 0F E6 2A 42 E6 2B F1 E6
6250 22 00 E6 23 09 A6 2C 40
6258 EB 03 E6 22 10 D6 43 00
6260 44 13 13 ED 42 D7 E6 23
6268 20 E4 08 2A E4 09 2B 26
6270 27 20 36 26 00 A6 2C 40
6278 EB 28 26 26 10 8B 23 A6
6280 13 02 EB 50 E6 22 00 E6
6288 23 00 0C 0F 1C E0 E4 08
6290 2A E4 09 2B 04 E1 2B 14
6298 E0 2A 24 E1 29 34 E0 28
62A0 7B 32 E4 28 24 E4 29 25
62A8 A6 24 00 EB 05 A6 25 00
62B0 6B 22 A4 26 28 7B 18 A4
62B8 29 27 FB 13 E4 26 24 E4
62C0 27 25 D6 43 00 44 13 13
62C8 EB 0D E6 13 03 8B 08 D6
62D0 43 00 8B 03 E6 13 06 E6
62D8 12 00 AF E4 2C 26 D0 26
62E0 E6 27 00 AF E4 04 28 E4
62E8 05 29 24 09 29 34 08 28
62F0 AF 8D 40 4C FF FF FF FF
62F8 FF FF 8D 40 50 8D 40 6B
6300 E6 2D 90 70 22 70 23 70
6308 24 70 25 D6 43 E5 D6 43
6310 C1 20 E5 EB 06 A0 22 80
6318 24 EB F0 50 25 50 24 50
6320 23 50 22 42 55 6B 04 E6
6328 13 01 AF E6 2D 80 A6 2C
6330 10 6B 03 E6 2D 90 E6 2E
6338 03 08 2A 18 2B 82 20 29
6340 2F D6 43 E5 28 2F 3C 08
6348 C0 E4 10 E2 10 E4 D6 43
6350 B3 46 E4 04 D6 43 B3 56
6358 E4 00 D6 43 B3 3A E9 4C
6360 30 A6 2C 10 6B 09 4C 40
6368 A6 2C 20 6B 02 4C 20 D6
6370 43 B3 46 E4 80 D6 43 B3
6378 2C 10 3C 00 D6 43 BA 76
6380 E4 02 6B 0D 80 E2 EB F4
6388 48 2D D6 43 B3 E6 13 05
6390 AF 48 2D D6 43 B3 D6 43
6398 C1 A4 2F E5 EB 0D A0 2A
63A0 A0 22 80 24 ED 43 36 E6
63A8 13 00 AF 00 2E EB 8A E6
63B0 13 02 AF 0C 70 1C 00 92
63B8 40 AF 0C 70 1C 00 82 40
63C0 AF 48 2D 46 E4 08 56 E4
63C8 EF D6 43 B3 2C 08 46 E4
63D0 04 D6 43 B3 48 2D D6 43

```

```

63D8 B3 D6 43 BA C0 E4 10 E5
63E0 48 2D 2A EA AF 28 22 38
63E8 23 A6 2C 10 EB 03 46 E2
63F0 08 A6 2C 40 7B 0F 48 E2
63F8 56 E4 00 90 E4 56 E2 07
6400 42 24 46 E2 08 48 2D 5C
6408 10 C0 E4 10 E3 10 E2 10
6410 E4 D6 43 B3 46 E4 02 D6
6418 43 B3 56 E4 FD D6 43 B3
6420 5A E7 AF

```


Z8SYS

Z8 SYSTEM SOFTWARE VERSION 1

ARNE THAGE 1984.11.04

INDEHOLDER:

EXTERNE I/O DRIVERE
LCD DISPLAY
CENTRONICS
4*4 KEYBOARD

EPROM PROGRAMMER
INITIALISERING

KODERNE ANBRINGES I EPROM FRA
ADRESSE 0000 OG FREMEFTER.
(HVLKET BETYDER AT DU SKAL
TRAEKKE 6000H FRA ADRESSERNE
I NEDENSTAAENDE LISTE)

ADR. ----- DATA -----
.....

6000 8F 56 FB F7 B0 21 56 15
6008 0F 6B 06 46 FB 08 44 15
6010 21 56 13 0F 6B 08 46 FB
6018 08 F0 13 44 13 21 0C 10
6020 1C 09 2C 40 3C 3C C2 42
6028 A6 E4 FE 6B 0D A6 E4 FF
6030 6B 02 92 40 A0 E0 A0 E2
6038 8B EC 9F AF 8D 40 4C FF
6040 FF FF FF FF FF 8D 40 50
6048 8D 40 6B FE 46 21 08 BF
6050 66 21 01 EB 03 8D 41 4D
6058 66 21 02 EB 03 AF FF FF
6060 66 21 04 EB 03 AF FF FF
6068 8D 41 E9 66 21 10 EB 03
6070 8D 40 8E 66 21 20 EB 03
6078 8D 41 27 66 21 40 EB 03
6080 AF FF FF 66 21 80 EB 03
6088 AF FF FF 8D 41 FF 28 13
6090 A6 E2 08 6B 06 A6 E2 20
6098 FB 0F AF 2C 10 D6 40 B3
60A0 2C 20 D6 40 A9 2C 10 8B
60A8 0A D6 40 BD 0C 60 1C 01

60B0 92 20 AF D6 40 BD 0C 60
60B8 1C 00 92 20 AF 0C 60 1C
60C0 00 82 30 76 E3 80 EB F9
60C8 AF 2C C0 76 15 01 6B 02
60D0 2C 80 44 13 E2 D6 40 B3
60D8 AF 44 13 13 EB 06 2C 01
60E0 D6 40 B3 AF 76 13 01 6B
60E8 04 2C 80 8B 02 2C C0 D6
60F0 40 B3 4C 28 2C 20 D6 40
60F8 A9 4A FB AF 2C 08 76 15
6100 01 6B 03 46 E2 04 76 13
6108 01 6B 03 46 E2 02 D6 40
6110 B3 AF 2C 01 D6 40 B3 2C
6118 0E D6 40 B3 2C 38 D6 40
6120 B3 2C 06 D6 40 B3 AF D6
6128 41 3A E4 13 02 56 03 DF
6130 46 03 20 D6 41 3A E6 02
6138 00 AF 66 03 04 6B FB AF
6140 E6 02 00 E6 F6 00 E6 F7
6148 41 46 03 20 AF D6 41 7A
6150 66 E2 F0 EB F8 D6 41 71
6158 D6 41 7A D6 41 9E 44 13
6160 13 6B F5 D6 41 71 70 13
6168 D6 40 6B 50 13 E6 12 00
6170 AF 0C FF A0 E0 80 E0 0A
6178 FA AF 0C F0 1C FE 2C FF
6180 5C 04 D6 41 8A 52 23 5A
6188 F9 AF 82 30 46 E3 0F 66
6190 E3 F0 6B 07 48 E1 46 E4
6198 F0 52 34 90 E1 AF 0C 41
61A0 1C C8 C2 30 A2 32 6B 0B
61A8 A6 E3 00 6B 0C A0 E0 A0
61B0 E0 8B EF A0 E0 C2 20 8B
61B8 02 2C 00 29 13 AF D6 41
61C0 7A D6 41 9E E6 12 00 AF
61C8 EE 37 DE 38 BE 39 7E 08
61D0 ED 34 DD 35 BD 36 7D 2C
61D8 EB 31 DB 32 BB 33 7B 0D
61E0 E7 30 D7 41 B7 42 77 43
61E8 00 56 21 F7 66 21 08 EB
61F0 FB E4 F0 13 56 13 7F 70
61F8 13 D6 40 6B 50 13 AF 66
6200 FA 10 EB FB 56 FA EF E4
6208 13 F0 AF E4 14 08 E4 15
6210 09 E4 12 0A E4 12 0C E4
6218 12 0E AF 00 00 00 00 00
6220 E4 15 2C A6 2C 10 6B 10
6228 A6 2C 20 6B 0B A6 2C 40
6230 6B 06 E6 13 04 8D 42 D7
6238 D6 42 DB D6 42 E4 A6 13

FEJLKODER OG INTERAKTIV DEBUG

3.4.1	FEJL/ERRORs.	40
3.4.2	Interaktiv debug	41

PROGRAM AFVIKLINGSTID/MEMORYPLADS

3.5.1	Introduktion	42
3.5.2	Begrænsning af hukommelsesplads	42
3.5.3	Forøget hastighed ved programafvikling	42

HUKOMMELSEN OG DENS OPBYGNING

3.6.1	Hukommelsesstruktur	43
3.6.2	Initialisering og opstart	43
3.6.3	Program format	45
3.6.4	RAM-top	44
3.6.5	Pointer registre - RAM-system	44
3.6.6	Registersystem uden RAM	46
3.6.7	Hukommelse organisering	47

SPECIALRUTINER I EPROM

4.0	Initialisering og NEW	49
4.1	Lager konfiguration	50
4.2	Alternative I/O-enheder	50
4.3	Centronics printer	51
4.4	Intelligent LCD-DOT-MATRIX display	51
4.5	4x4 keyboard	52
4.6	8x11 keyboard	52
4.7	EPROM-brænder	52
4.8	EPROM-systemdump	54

1.0 FORMÅL

CXZ8-CPU er en l'kort microcomputer med indbygget Basic-oversætter, I/O-porte, seriel UART, Centronics printerudgang og keyboard scanner.

På flere punkter KAN man få computeren til at ligne en hjemmecomputer, hvilket dog vil være upraktisk og for dyrt.

CXZ8-CPU er til styring og kontrol eller » dedikerede » computeropgaver. Primært opgaver hvor computeren efter programmering skal udføre en eller flere bestemte opgaver af sig selv. Og selv kunne starte op med programafviklingen efter strømsvigt. Modsætningsvis hjemmecomputeren, der skal slette alt program ved initialisering/power-up og gøre klar til programmering, skal styringscomputeren hoppe ind i programmet straks og udføre en opgave uden afbrydelser. Ved opbygning af en fornuftig software kan styringscomputeren også bringes til at starte op med at regenerere de data et eventuel strømsvigt har medført. Ved ekstra hardware vil man ligeledes kunne måle forsynings-spændingen og lagre data inden de tabes for altid.

CXZ8-CPU kan også dedikeres til specielle dataopgaver. Det kan være mindst ligeså interessant, som styringsopgaverne. Da du kan tage imod og sende serielle data, oplagre typisk 8K (max 28Kbyte's), skrive direkte på en Centronics-parallell printer, scanne et keyboard, og senere også tilslutte en videogenerator, kan du bygge elektronisk skrivemaskine, dataterminal, dataopsamler og netvært til max. 19.200 BAUD og intelligente måleinstrumenter. Mange professionelle måleinstrumenter har dataudgange/indgange for RS232. De kan tilsluttes over et niveauskift til computeren. Eller du kan opbygge måleinstrumenter MED SELVE Z8-computeren. Bare et måleinstrument til effekt, kan være vanvittigt dyrt at lave i analog teknik. Udført med en processor kan du få måleresultater på ethvert matematisk beregningsgrundlag direkte på skærmen. Og du kan vælge måleområde fra computerens keyboard. Hvad giver en forstærker f.eks. samtidig i effekt på 2 kanaler? Du skriver belastningsmodstand og computeren udregner effekten efter ohm's lov. Eller du lader den måle strøm og spænding, så den kan vise effekt og belastningsimpedans samtidig på søjle, kurve eller i digitale tal.

En datapræget opgave for CXZ8-CPU kunne være en databuffer mellem et antal skoledatamater og en eller flere printere. Eller opbygningen af et lokalt netværk til 19.200 baud. Det ville kræve en processor til terminalen og en på hovedcomputeren, som kunne tage sig af opsamling og udveksling af data til den tid hvor hovedcomputeren er klar.

Circuit Design lancerer i det kommende år en række konstruktioner til CXZ8, som kan en række forskellige ting. Måske ikke lige DET du mangler nu, men sikkert noget du kan kombinere dig frem til. For computeren programmeres fra en RS232-terminal eller en billig hjemmecomputer i BASIC! Det gør det hurtigt og nemt for folk med blot lidt BASIC-erfaring at udvikle computer-systemer selvstændigt. Vi har koncentreret os om at bruge Spectrum, VIC20 og CBM64 som udviklingsterminaler til opgaven.

Circuit Design bringer softwaretips i samtlige medlemsblade i 1985 til CXZ8-CPU under betegnelsen »Arnes Hjørne» - det er de 4 midtersider i »Circuit».

I tilslutning til det vil de kommende CD-computerkonstruktioner blive konstrueret således at de BÅDE kan benyttes til Z8'eren og de normale hjemmecomputere vi nu som tidligere har udviklet konstruktioner til: CBM64, Spectrum, ZX81 og VIC20. Vi håber også i 1985 at kunne introducere konstruktioner parallelt til CP/M, MTX og PC/IBM'er.

EPROM'er af typen 2716/2732 og 2764 kan brændes i Circuit Design's hardware konstruktion CX81-PRM og 2764-pirntet CX81-64.

Du skal altså IKKE nødvendigvis bruge din egen udviklingscomputer, et specielt program eller en stor EPROM-brænder. Bare plug PRM-brænderen i 20-pin usersoklen på Z8-computeren. Så klarer den resten.

Svar og fejlmeldinger sendes ud til til terminalen.

Kald af EPROM-rutinen:

PRINT USR (%4220, (type-eprom),(eprom-nummer)

(type-eprom) er tallene 16, 32 eller 64, som igen afhænger af den benyttede EPROM-type: 2716, 2732 eller 2764.

(eprom-nummer) er tallet 1 eller 2. Hvis dit program er for stort vil det kunne deles op i to EPROM'er. Første skal da have tallet 1 og en eventuel anden tallet 2. Se fejlrapportkode 3.

Når brændingen er slut, eller hvis en fejl detekteres, udskrives en af følgende fejlkoder:

- 0: EPROM'en har taget programmet ind og er klar
- 1: EPROM'en ikke slettet og kan ikke programmeres
- 2: Programmeringsfejl (defekt EPROM)
- 3: EPROM'en er fyldt op. Brænd endnu en EPROM ved at vælge (eprom-nummer) 2, eller vælg en større EPROM og start igen.
- 4: (eprom-type) nummeret er valgt forkert. Er ikke et af de 3 mulige tal 16, 32 eller 64.
- 5: Time-out fejl i brænderen. Check kredsløbet CX81-PRM - er der f.eks. sat netspænding til (mest normale glemme fejl).
- 6: (eprom-nummer) er ikke 1 eller 2, eller (eprom-nummer) er 2 selv om Basic-programmet GODT kan være i en 4K EPROM.

KOMMENTARER TIL BRÆNDINGEN

Hvis programmet fylder mere end 2K i en 2716, må den første ikke være en 2716. Du må vælge en 2732 eller 2764. Den sidste kan dog på grund af processorens arkitektur IKKE benyttes i de nederste 4K. Men det vil være muligt at lægge MC-koder i hexområdet %800 til %FFF.

EPROM'erne kan mixes efter ønske, men du kan maksimalt plumpe 12K EPROM i 2 sokler. F.eks. 1-1/2 2764 eller en 2732 og en 2764.

Først brændte EPROM sættes i soklen IC2 og anden sættes i soklen IC3.

EPROM-softwaren benytter de interne Z8 CPU-registre 34-47 (interne variable fra A til N).

FLYT cursor Skriver du instruktionen GO @ %40C9 , (line) , (col) sættes cursoren på linien (enten 1 eller 2) og kolonnen 0-23 (39). Da du normalt kun viser 24 af de 40 karakterer vil cursorset på »col» over 23 ikke ses før man flytter rundt på linien.

CLEAR display ON/OFF display Instruktionen GO @ % 40D9 , (line) indsætter blanke tegn i hele linien. Brug (linie) = 1, 2 eller 0, som sletter hele displayet.

Du kan tænde og slukke displayet med GO @ % 40FC, (disp-on) , (cur-on) vil tænde og slukke henholdsvis displayets karakterer og cursor. Sætter du tallet »1» ind er displayet ON. Med »0» slukker du henholdsvis display og cursor. Du skiller med et komma.

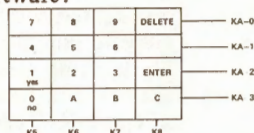
AVANCERET DISPLAY UDNUTTELSE

Yderligere displayfunktioner, som SCROLL, brugerdefinerede tegn (ÆØÅ etc.) m.v., kan opnås ved direkte udlæsning til kontrolporten » @ % 6000» og dataporten » @ % 6001». Men de data man skal indlæse er ikke altid lige nemme at finde rundt i. Det kræver indsigt og forståelse i datateknik. Derfor tillader vi os her at henvise til fabrikantens datablad. Evt. fotokopi vil kunne fremskaffes fra Circuit Design mod frankeret svarkuvert (kr. 5.50) og kr. 10 i frimærker.

4.5 MATRIXKEYBOARD 4x4:

USR (%41BE)

I masser af tilfælde vil en styring kræve et simpelt keyboard. Ikke blot en indtastning fra en port. Derfor har vi udviklet en keyboardscanning software rutine for 16 taster. Tasterne forestiller vi os monteret til adresseledninger på den 20-pol'ede konnektor's ben KA0-3 og K5-6-7-8. Du kan få tallene 0-9, ABC og et par funktionstaster som ENTER, DELETE, KOMMA og evt. NO/YES med. Her har du vort bud på hardware til den indbyggede software:



Indlæsningskommandoerne er beskrevet under 4.2 Alternative Input/Output Enheder.

INKEY Funktionen USR (% 41BE) returnerer ASCII-koden for en nedtrykket tast. Hvis ingen tast nedtrykkes returneres et »0». Det samme gælder hvis du nedtrykker flere taster samtidig.

Funktionen gælder SELVFØLGELIG KUN et 4x4-keyboard - ikke RS232.

4.6 ASCII-KEYBOARD

Vi har endnu ikke implementeret software for et fuldt ASCII-keyboard. Det sker først i 1985. Men det ER altså planlagt og benyttes i terminalapplications. Hvis du f.eks. vil udvikle en intelligent terminal med RS232-udgang er CXZ8 med ASCII-software sagen. Du kan taste ind på som en normal skrivemaskine, få det ud på en printer, skærm eller modem. Du skulle faktisk også kunne lave din egen telex, testapparat, kommandocentral m.m.

4.7 EPROM BRÆNDER/ PRINT USR (%4220, (type eprom),(eprom.nr)

Som du nu bør have erfaret har vi udviklet en system-EPROM med en række af de features du mangler i selve Z8-ROM'en. Bl.a. også en EPROM-brænder rutine, så din computer kan dublikere sig selv.

Circuit Design har planlagt hard- og software til et computerstyret » tænd-sluk-ur », en styring til varme, belysning eller maskiner, voltmeterkredsløb, mødeur til personale - med sammenregning, dataopsamler, databuffer, telexterminal og modemterminal.

1.2 TEKNISKE MULIGHEDER / SPECIFIKATIONER

CXZ8-CPU er opbygget med flere funktioner på samme printkort (1/2-europa-kort størrelse). Z8-kortet og de kommende ekstrakonstruktioner vil kunne placeres i den lille flade B2015 aluminium indbygningskasse på 15x11x4cm. Skal du KUN bruge grundcomputeren kan du evt. klare dig med den lille B2010 på 10cm. Lad os se på CPU-kortets muligheder:

SPECIFIKATIONER

CPU: Z8671 fra ZILOG med indbygget Basic/Debug ROM. En C-MOS udgave er annonceret til 1985 fra Zilog's side.

Memory: 3 sokler til 2, 4 eller 8Kbyte statiske kredse - valgfrit RAM eller EPROM-hukommelser. Eksempel: IC2: Circuit-Design system-EPROM: PZ8-S1, din egen BASIC EPROM, en 2716/32/64 og endelig en 2K eller 8K statisk RAM type 6116 eller 6264.

Display:: Sokkel for fladkabeltilslutning af standard 14-pin intelligent LCD-display - primært 24-40chr a' 2 linier: OPTOREX /SANYO (CD: H-LCDDOT).

UART: Indbygget duplex RS232/5V ind- og udgang med variabel baud-rate fra 110 til 19.200 baud i standard spring.

IN/OUT: 8 bit ind og 8 bit ud på 18-leder fladkabel med plus 5V/Gnd. Standard for CD-interface.

Centronics: Printerudgang for Centronic's parallelttilslutning med 8 data og strobesignal ud, samt busysignal ind. Tilsluttes via D-CENTRON fladkabel umiddelbart. Ellers universel 8 bit OUT-port.

Expansion: Samtlige signaler på 34-pol konnektor for udbygninger. Har adresseledninger A0-15, data D0-7, DS, RD/WR, RESET, Timer in, Timer out, Interrupt-1, Interrupt-2, plus-5V og Gnd. DS, WR og RESET er aktiv low. Andre aktiv high. Soft restart mulig ved samtidig tilslutning af Serial IN og Reset til Gnd.

1.3 UDVIDELSESMULIGHEDER

Circuit Design har planlagt følgende projekter til Z8-computeren:

- 1) 8-bit netstyring for TRIAC tænd/sluk af belysning/lysshow etc.
- 2) Kort med RAM-udvidelse for 2x8K statisk med 6264.
- 3) Kort med ekstra I/O-porte for PLC-styringer (til pkt.1 også).
- 4) AD-converter for måling.
- 5) DA-converter for referencespændinger.
- 6) Steppermotor driver for » skildpadde » eller robot styring.
- 7) Digitizer input board.
- 8) VDU-kort til display på monokrom eller farve monitor (ikke TV).
- 9) Terminalprogram for lokaltelex.
- 10) Seriel dataopsamler for netværk med seriel kommunikation.

2.0 OPBYGNING

CXZ8-CPU er opbygget på et lille printkort i 1/2-Europa-størrelse. Kortet har dog ikke Europa-standard stik, fordi det ikke er tanken at opbygge en multi-kort computer. Derfor er alle stik ført ud fra printkortets 2 sider. Den ene har CD-bus'en for udbygning af RAM, I/O og specialfunktioner, den anden har bøsninger for 20-pin I/O, 14-pin for intelligent dot-matrix LCD-display, seriell RS232/5V port (ikke standard - blot fladkabel) og forsyningsterminaler.

Keyboard kan evt. forbindes med fladkabel til et specialstik midt på printet. Hvis du ikke ønsker intelligent keyboard benyttes I/O-porten.

Samtlige IC-komponenter på Z8'eren monteres i sokler. Man kan nemlig benytte mange memorykombinationer efter formålet.

Det er absolut IKKE NØDVENDIGT at montere SAMTLIGE IC-kredse. Hvis du f.eks. ikke har brug for I/O-port monterer du ikke de IC-kredse der hører til denne port.

2.1 MEMORY ORGANISERING

Z8671-processoren undersøger under opstart hvor hvilken baud-hastighed man vil kommunikere med og hvilken memory konfiguration man har monteret. Vi har UDVIDET en del af kredsløbets faciliteter med en speciel SYSTEM-EPROM, som du bør isætte. Den indeholder ekstra software for 4x4 keyboard-funktion, Centronics-printer driver og MEMORY-organisering. Derfor kan du med CD's computer benytte blandede EPROM'er og RAM'er fra Basic'en.

Z8 er lidt speciel i memory-henseende. Der har det bedst med hukommelse i blokke af 1, 2 eller 4K bytes. Men da processoren blev skabt - omkring 1980-81, var større hukommelser ganske kostbare. Idag er 8K og 16Kbyte's » normale » og koster mindre end de små eller det samme. Derfor lavede vi vor Z8-udgave til 2-4-8K RAM og EPROM. Men hvis du bruger en 8K som system EPROM i bunden af maskinen må du desværre undlade halvdelen, - dvs. de nederste 4K. Det er og kan kun være spildt, hvis man ikke skal bruge for flere kroner logik-kredse end spildt EPROM. Sætter du 2 stk. 8K-EPROM i Z8 får du kun 4K i bunden, men selvfølgelig 8K i toppen.

MEMORY-søjlen på tegningen fig.2.1. viser dig hvor forskellige størrelser hukommelse mappes ind af adressedekoderne. Ordet MAP er det engelske udtryk for KORT eller PLAN. Deraf det afledte memory-mapping.

CXZ8-CPU'en er ikke fuldkommen dekodet. Derfor må du finde dig i at I/O-porte og KEYBOARD-scan mv. fylder op imod 1K i dekningen. Du kan altså ikke adressere den fulde hukommelsesstak, hvilket næsten heller aldrig er nødvendigt. Vi mener med vor Z8-memory-map at have skabt en computer med adressering, som kan dække 90% af alle formål. Andet må man adressere sig til selv over CD-bus'en.

ADRESSERINGSJUMPERE

For enden af hver memory-IC har du en 5-pol-jumper. Den skal monteres afhængig af hvilken type hukommelse du bruger.

Printpladen CXZ8-CPU forbinder ikke benene nr. 27 på IC2, 3 eller 4 nogen steder hen. Hvis du bruger 28-pin kredse skal 27 loddes til plus eller pin 5 på jumpermarkeringen for hver af IC'erne 2, 3 og 4.

LIST af et Basic program på Centronics-printer:

```
GO @ % 4000 , 0 , 2
LIST
GO @ % 4000 , 0 , 0
```

hvis du i stedet ønsker at LIST'e programmet uden at få skrevet ordet LIST på printerens, kan du indlægge listningen i en programstump først:

```
1 GO @ % 4000 , 0 , 2
2 LIST 5 , 9999
3 GO @ % 4000 , 0 , 0
4 STOP
```

skriv derefter RUN

BEMÆRK: CPU-register 33 må ikke ændres mens (in-unit) eller (out-unit) forskellig fra 0 kører.

4.3 CENTRONICS PRINTER

GO @ 4140

Vor Z8-software er udlagt således, at den kan drive en Centronics-printer direkte. Porten findes også på CXZ8-CPU-printet, så det eneste man skal gøre er at slutte det rigtige printer flad-kabel (CD-Medlems-Service: D-CENTRON). Før man kan skrive til printerens skal driverprogrammet initialiseres. Det sker ved kald af rutinen GO @ % 4140.

Herefter kan der PRINT'es eller LIST'es som beskrevet under punkt 4.2.

Husk, at de fleste printere skal have en LF/CR for at skrive hvis liniebufferen ikke fyldes. Skriver du blot » A », sker der altså ikke noget før » A » efterfølges af en CHR10 eller CHR13. Det sker dog automatisk ved kommandoen PRINT.

4.4 SPECIELLE RUTINER TIL STYRING AF LCD-DISPLAY

CXZ8-CPU'en er designet med porte for drift af et standard LCD-dot-matrix display med 2 linier på hver 24 karakterer. Displayet er standard hos flere - vi benyttede OPTOREX DMC 24227 (Nordisk Elektronik A/S) og STANLEY GDM2401 (Ditz Schweitzer A/S). Denne type display ser ud til at blive normal industristandard indenfor intelligente displays.

LCD-dot-matrix displays kan vise næsten alle tegn opbygget af 5x7 prikker og en cursorstreg.

Som display under programmering er de to linier kun en nødløsning.

Men som display til maskinstyringer med X/Y/Z-angivelse, som display for avancerede synthesesmodtagere med frekvens, station og kanal, som display for alarmanlæg eller adgangskontrol er de 2 linier af 24 karakterer perfekt. Da strømforbruget samtidig ligger på omkring 1mA, er det også fint til batteridrevet og mobilt udstyr. Specielt når CXZ8-CPU'en i løbet af 1985 vil komme i en C-MOS udgave, - ligesom RAM'er og EPROM'er allerede nu kan leveres.

MEGET VIGTIGT: INITIALISERING

GO @ % Kommandoen GO @ % 4112 initialiserer displayet:

4112 2 linier tal og tekst af hver 40 tegn. Deraf vises altid de første 24 tegn. Cursoren flyttes til højre under skrivning. Displayet og cursoren er ON. Cursoren sættes i begyndelsen af linie 1 displayet CLEAR'es.

efter initialiseringen. Ellers ved Z8 ikke hvor den må lægge Basic-programmet. Initialiseringen er selvfølgelig IKKE nødvendig hvis du kører et automatisk EPROM-program, for vores EPROM-brænder rutine lægges selv initialiseringen ind i starten af EPROM'en.

Hvis et EPROM-program laves endeligt (under udvikling af det) - modsat loopet til automatisk afvikling - kan initialiseringsrutinen anvendes til at overflytte kontrollen til RAM-lageret. Man kommer retur til EPROM'en ved at skrive GO @ % 420B, %1020, (RAM-slut adresse) og RUN.

4.1 LAGER KONFIGURATION

Memorysoklerne IC2 og IC3 anvendes normalt til brugerprogrammet i enten RAM eller EPROM. IC4-soklen benyttes normalt til system software (maskinkoderutiner. CD's EPROM-program med systemsoftware skal altid sættes i IC4. Det er den software, som giver ekstrafunktionerne som f.eks. EPROM-brænder. Soklerne kan monteres med 2K eller 8K RAM kredse eller 2, 4, eller 8 K BYTES EPROM. Husk at vi her snakker om bytes - ikke bit. EPROM'erne er derfor af typen 2716/32/128.

Til udviklingsarbejde anbefales RAM i IC2-soklen og måske også i IC3-soklen. BASIC-programmer i EPROM skal starte i IC2-soklen hvis programmet skal starte op automatisk når computeren tændes.

Om nødvendigt kan programmet fortsætte i IC3-soklen. Eller man kan forsyne IC3-soklen med RAM til datalager.

Hvis du vil bruge vores alternative Input/Output styringer, SKAL der være en BASIC EPROM eller en RAM-kreds i IC2-soklen. Hvis der er RAM i IC2-soklen skal du huske at køre den efterfølgende initialiseringsrutine efter systemopstarten.

VIGTIGT: Hvis du benytter 8K EPROM'er (2764) SKAL ben 27 kortsluttes til ben 28.

4.2 ALTERNATIVE INPUT/OUTPUT ENHEDER

Ved hjælp af et sæt maskinkodeinstruktioner er det muligt at få Z8'ens PRINT og INPUT/IN-instruktioner til at læse fra og skrive til andre ydre enheder end RS232-porten.

Omskiftningen sker mellem I/O-enheder ved kald af I/O-kontrolrutinen:

GO @ % 4000 , (in-unit) , (out-unit)

(in-unit) er en af følgende:

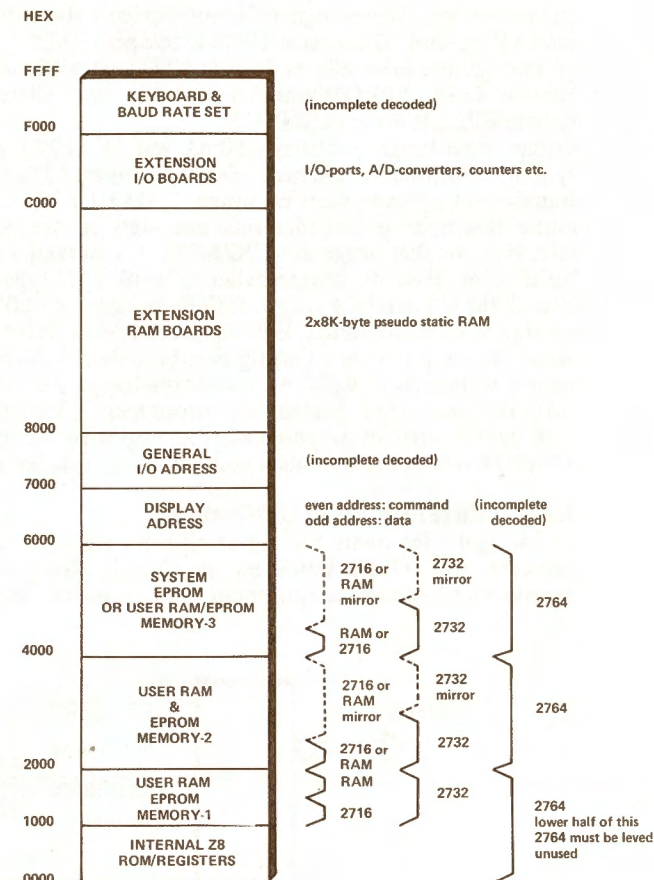
- 0: Input fra RS232
- 1: Input fra 4x4 keyboard (se speciel beskrivelse)
- (2: Input fra 8x11 keyboard - implementeres senere)

(out-unit) er en af følgende:

- 0: Output til RS232
- 1: Output til LCD-display (se speciel beskrivelse)
- 2: Output til Centronics-printer.

CXZ8-CPU
vers.2

MEMORY MAP
Arne Thage



- 2716 En 2K-EPROM med 24 ben. Bruger du den skal den i de 24 huller nærmest printpladens kant. Desuden skal dens jumper forbindes fra 3 til 4.
- 2732 Er en 4K-EPROM med 24 ben. Den skal adresseres på A11. Det sker ved at forbinde dens jumperfelt fra 1 til 2. IC'en monteres nærmest printets kant.
- 6116 Er en 2K-RAM. Den har R/W-indgang på ben 21 (23 iflg. 28-pin). Derfor skal du sætte en forbindelse i jumperfeltet fra 4 til 5.
- 2764 Er en 8K-EPROM med 28 ben. I IC2-soklen kan du jfr. teksten kun køre den på de åverste 4K. Du forbinder en lus bag på printet fra pin 27 til 28 og sætter en forbindelse i dens jumperfelt fra 1 til 2. Det adresserer A11.
- 6264 Er en 8K statisk RAM, som ligner 2764. Den har WE på (R/W) pin 27. Derfor må du forbinde en tråd på printets bagside fra pin-27 til dens jumperfelt nr.5. Samtidig skal du sætte jumper i 1 til 2.

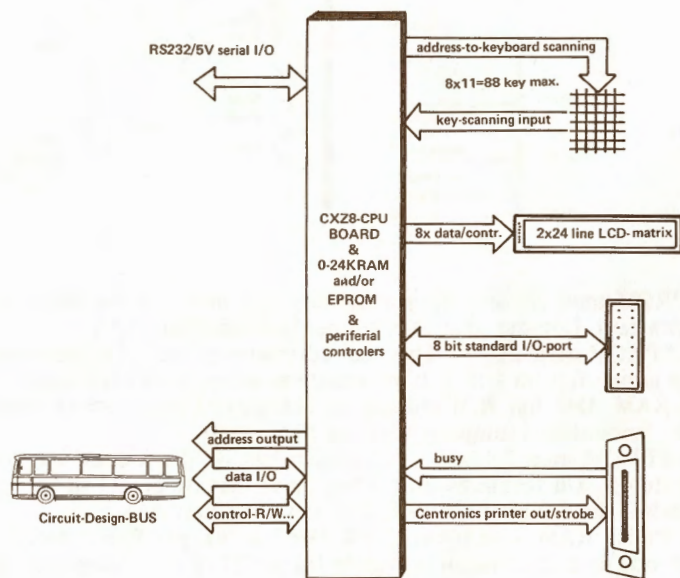
Centronicsudgangen kører direkte på CPU'en. Der er 8 portudgange, strobe udgang og busy-indgang. Hver af disse udgange klarer som alle de andre 3-5mA eller 2 LSTTL'er på hver.

I/O-porten på Z8'eren ligner Circuit Design's standard. =-bit ind og 8 ud, samt plus 5V og Gnd. Tilsammen 18-20 ledninger (19-20 benyttes ikke). Med porten vil man kunne drive alle de kendte CD'konstruktioner med det rette program. Således f.eks. EPROM'brænder rutinen, som allerede ER implementeret i system-EPROM'en - se afsnit 4.7.

Computeren bruger omkring 350mA ved 5V (10%) og den eksterne strømfor- syning er forholdsvis ukritisk. Du kan benytte CD's miniforsyning med T2605 transformator/kasse, samt en simpel 7805 eller LM317. Vi har tidligere vist en masse eksempler på hvordan man gør »det», så her lader vi det være op til dig selv. Hvis du skal bruge din CXZ8-CPU » i marken » kan du forsyne den fra 4 NiCd-celler. Hvis du bruger celler af 1 til 4AH-typen kan du lade med I/20, eller 200mA i uendelig lang tid. Cellerne tager de 200mA uden at blive skadet og afgiver blot lidt varme. Hvis din ladestrøm er 550mA kan du holde liv i både celler og computer og vil aldrig få afbrydelser. I dette tilfælde vil en strømfor- syning så begrænse sig til en transformator på 5-6 volt, en ensretterbro og en lademodstand, som bestemmer strømmen. Strømmen kan beregnes eller bare prøves med et amperemeter og nogen forskellige modstande i området 4,7-100 ohm. Det er nemmere end det lyder - selv for en datalog !

2.3 PERIFERE TILSLUTNINGER

Vi har gjort det nemt for dig at tilslutte de ydre enheder. Alt kan ske med fladkabel og MOLEX-bøsninger på 10, 14, 20 og 34 poler. Det er nemt og overskueligt. Centronics-printeren kobles til ved blot et montere et kabel.



Hvis GOSUB-stak'en når dette tal er resultatet overflow. NEW-kommandoen sætter disse registre til R8-R9's tal plus stakreserven. Et tal bestemt af den 5'te byte i konstantblokken. Efterhånden som programlinierne indtastes vil disse registre hoppe frem i samme takt så linierne beskyttes. Samtidig beskyttes dine programlinier af stakreserven så error's ikke ødelægger noget tidligere skrevet. Z8671 I/O-porte.

00-03 0-3

SPECIALRUTINER I EPROM

4.0 INITIALISERING GO @ 420B

GO @ 420B, (RAM-start adresse),(RAM-slut adresse) efterfulgt af NEW.

På grund af den ufuldkomne dekodning af 2K-RAM'er er det nødvendigt at køre en særlig initialiseringsrutine efter systemopstart for at sætte Z8'ens memory pointere på plads. Ellers tror computeren, at der er meget mere RAM-plads til rådighed end i virkeligheden. Hvis du benytter de forholdsvis nye 8K statiske RAM'er behøver du kun køre initialiseringsrutinen, hvis du har RAM i soklen IC2 og du vil bruge de alternative I/O-rutiner.

Initialiseringsrutinen skaber nemlig plads til nogle meget nødvendige pointere i begyndelsen af RAM-lageret. Hvis du bare går igang med at programmere Basic UDEN initialiseringen overskrives pointerne. Det ser man ikke resultatet af før det går galt. Har du således indskrevet et par tusind bytes uden initialisering kan det hele være spildt og skaden er uoprettelig !

Nedenstående skema viser hvilke værdier der skal indsættes som start og slut-adresse for de mest almindelige memory konfigurationer:

IC2-sokkel	IC3-sokkel	IC4-sokkel (1)	(start adresse)	(Slut-adresse)
2K RAM	— (3)	—	%1820	%1FFF
—	2K RAM	—	%2000	%27FF
—	—	2K RAM	%4000	%47FF
2K RAM	2K RAM	—	%1820	%27FF
—	2K RAM	2K RAM	%3800	%47FF
2K RAM	2K RAM	2K RAM	%1820	%27FF (2)
2K RAM	—	2K RAM	%1820	%1FFF (2)
8K RAM	—	—	%1020	%1FFF
8K RAM	8K RAM	—	%1020	%3FFF
—	8K RAM	—	%2000	%3FFF
8K RAM	8K RAM	8K RAM	%1020	%5FFF

- (1) Denne memory skal normalt køre med system EPROM'en.
- (2) RAM 3 kan kun benyttes som DATA-lager i denne konfiguration.
- (3) — viser at denne memory-sokkel er tom eller indeholder EPROM

VIGTIGT HUSK:

Z8'eren SKAL initialiseres til vor RAM-opbygning og du SKAL skrive NEW

REGISTER FIL

00FF	255	Stak pointer (bit 7-0)
00FE	254	Stak pointer (bit 15-8)
00FD	253	Register pointer
00FC	252	Program kontrol flag
00FB	251	Interrupt maske register
00FA	250	Interrupt request register
00F9	249	Interrupt prioritet register
00F8	248	Port 0-1 mode
00F7	247	Port 3 mode
00F6	246	Port 2 mode
00F5	245	T0 prescaler
00F4	244	Timer/counter 0
00F3	243	T1 prescaler
00F2	242	Timer/counter 1
00F1	241	Timer mode
00F0	240	Serial I/O
80-EF	128-239	Ingen registre
40-7F	64-127	Regneudtryk stak. Bruges af Basic/Debug
21-3F	33-64	Frie registre. Benyttes ikke af Basic/Debug
20	32	Cursor position for udprintning
1F	31	Her gemmer oversætteren interne variable
1E	30	Basic/Debug oversætter variable
1C-1D	28-29	Konstant blok pointer
18-1B	24-27	Basic/Debug interne variable
16-17	22-23	Linienummer benyttet lige nu
14-15	20-21	Andet argument i USR-subrutine kald ved 3-argument.
12-13	18-19	Sidste argument i USR-kald og resultat af kald
10-11	16-17	Benyttes af Basic/Debug
0E-0F	14-15	Pointer for næste karakter i inputlinie bufferen
0C-0D	12-13	Pointer til liniebuffer enden. Indeholder sidste karakter du har lagt i liniebufferen i direkte mode eller sidste karakter i forhold til "?" retur fra CPU til terminal. R12 (0C) indeholder tallet for den højeste RAM-side eller 00 hvis der ikke er RAM.
0A-0B	10-11	Register R10-R11. Pointer til stak-bund, højeste adresse på brugerens RAM. Ved opstart får stakpointeren den værdi denne pointer har. Den indstilles til xx20 af den højst tilgængelige RAM.
08-09	8-9	Register R8-R9. Indeholder første adresse for brugerens program og samtidig laveste byte af RAM-hukommelsen. Hvis der er ekstern EPROM er adressen 1020 (HEX).
06-07	6-7	Register 6-7 markerer GOSUB-staken og maskinekodens begyndelse. Den skubbes 2 byte ned per GOSUB og hopper 2 byte op for hver RETURN. Hvis denne pointer krydser program-slut pointeren får du overflow, og Basic/Debug hopper ud af programmet eller tilbage til sidste statement. Programstaken i slutningen af programmet sikrer dig imod ødelæggelse af koden.
04-05	4-5	NEW og STOP nulstiller denne pointer til R10-R11's tal. Register 4-5 viser RAMTOP for Basic'en plus stakreserven.

Alt passer sammen. Det samme gælder I/O-port og LCD-dot matrix display. Derimod varierer terminaltilslutningen en del, idet forskellige terminaler har forskellige tilslutninger. Du kan således IKKE køre over en rigtig RS232-terminal, fordi de logiske niveauer skifter fra plus til minus med niveauer mellem 3-15V. Differenspændingen kan være 6 til 30V. Du skal altså bruge en LOGIK RS232-udgang, som på VIC20/CBM64 direkte. Eller en port CX81-I/O, som på Spectrum. Derimod IKKE Interface-1 fra Spectrum, som HAR levelshift fra plus til minus.

Seriell udgang på CXZ8-CPU bøsning B4 pin-8 sættes til Seriell ind på f.eks. VIC20/C64 Seriell indgang B med pin-2 eller 7 på B4 til Gnd. på VIC20/CBM ben N. Serial out på VIC20/CBM er pin-M. Den skal sende keyboardets kode til serial-in pin-9 på Z8'eren. OUT på Z8 skal til IN på terminalen. OUT fra terminalen skal til IN på Z8. Bruger du I/O-interface CX81-I/O eller CEN fra en Spectrum-48 til udviklingsterminal skal soklen's pin-1 til Z8's pin-8 på B4. Pin-9 på CX81-I/O-soklen skal til 2 på Z8 og pin-17 på CX81-I/O-soklen til pin 9 på Z8. Det lyder svært når det skal skrives men er nemt iflg. diagrammet.

2.2 DIAGRAM

Z8671/IC1 er selvfølgelig hjertet i computeren. Den har indbygget en masse funktioner, som gør det muligt for os at lave en så enkel og ydedygtig konstruktion som CXZ8-CPU. Vi har ikke her mulighed for at gengive de 2-3.000 sider dokumentation over Z8 på engelsk, som Zilog-producenter har skrevet. Dem kan du dog købe i boghandlen/Medlems-Service: LZ-VOL-1-2, LZ8-ASS, LZ8-BI og LZ8M til mellem 100-250 kroner per bog! desværre.

Z8'eren skal have et krystal på 7MHz. Desuden en forsyningsspænding på 5V. Resten af benene går til I/O-porte og data/adresse-ledninger. For at spare ben har man endvidere multiplexet dataudgangene med adresserne A0-7. De de-multiplexes af IC8 til drift af IC2-3-4, som kan være RAM eller EPROM'er. Memory og porte adresseres på plads efter den » memory-mappe » vi viste tidligere med dekodeerne IC9/10 og nogen LS04-invertere.

Keyboardscanning tages som krydsmatrix mellem de 11 adresseledninger og dioder på KA0-10 samt de dekode dataledninger D3-7 på IC7. Data fra D0-D2 benyttes til baud-hastigheds indstilling under opstart. Processoren starter nemlig med at aflæse denne addresses data under opstart. Tallet bestemmer hvilken UART-hastighed processoren skal sende og modtage på Serial-in (5) og Serial-out (4). UARTEN er jo OGSÅ indbygget i processoren.

2.4.0 TERMINALPROGRAMMER

Vi har udviklet programmer til et par små systemer for Z8-BASIC design til Sinclair-Spectrum og CBM-hjemmedatamaterne. De loades med "PS-Z8" for Spectrum og PC64-Z8 for Commodore 64. Men du kan også lave dit eget program på enhver anden datamat med RS232-udgang i BASIC efter Spectrum-programmet vi viser andet sted i denne beskrivelse.

I det efterfølgende har du også et miniprogram til VIC20, som gør den til en terminal på userporten. Samme program kan også køre på CBM64, men selvfølgelig IKKE med alle de lækre faciliteter.

2.4.0.1 BAUD-RATE INDSTILLING

Du skal indstille Z8'ens indbyggede serielle UART til den baudhastighed du anvender fra terminalen. I de fleste tilfælde kan du køre 1.200 baud fra termi-

nalen, så vi går ud fra at du ønsker denne indstilling. Men Z8'eren undersøger ved opstart BAUD-RATE indstillingen på nogle » jumpere » ved J1.

Den er også mærket BAUD-RATE og tallene 1-5. for hvert printhul. hullerne 2 og 4 går til minus. 1, 3 og 5 går til de adresserede dataindgange LSB, NSB og MSB - det betyder Least-Significant-Bit, Next-Significant-Bit og Most-Significant-Bit - altså: mindst, næstmindst og mest betydende bit.

Her har du en tabel du kan benytte ved programmering af BAUD-RATE:

110 BAUD	Forbind:	4-5
150 BAUD	Forbind:	1-2-3-4-5
300 BAUD	Forbind:	ingen
1.200 BAUD	Forbind:	2-3
2.400 BAUD	Forbind:	2-3 og 4-5
4.800 BAUD	Forbind:	1-2
9.600 BAUD	Forbind:	1-2 og 4-5
19.200 BAUD	Frobind:	1-2 og 3-4

Som du kan se er der mange muligheder ud over de sædvanlige 1.200 baud du kører under programmeringen. Hvis du f.eks. vil bygge computeren til et netværk af terminaler kan du programmere en til hver terminal og en til hovedterminalen. Derefter stiller du til 19.200 baud og kobler RS232'en fra den ene til den anden. Terminalenheden kan så fungere som kasseapparat(er) til den Z8, som modtager. Den skal så gemme data fra terminalen i sin RAM og hovedcomputeren kan så hente data på forespørgsel. En slags multitasking netværk i lille skala. Som du vil se senere har vi også tænkt på at udvikle en MC-rutine til et stort terminalkeyboard, samt en VDU-driver for skærm. Så har du en intelligent seriel 19.200 baud terminal ! Hvor DU bestemmer programafviklingen !

2.4.0.2 SKÆRMBILLEDE FOR UDVIKLINGSPROGRAM

Det skærmbillede du ser efter load af programmet til CBM64 eller Spectrum skulle gerne se således ud:

Z8-BASIC DEVELOPMENT SYSTEM CD

MENU:

T: RS232 TERMINAL

G: GET Z8 PROGRAM INTO MEMORY

P: PUT MEMORY PROGRAM TO Z8

H: HARDCOPY MEMORY PROGRAM

S: SAVE MEMORY PROGRAM ON TAPE

L: LOAD MEMORY PROGRAM FROM TAPE

SYSTEM BESKRIVELSE

PS-Z8 udviklingssystemet er udformet af Ing.Arne Thage specielt til BASIC-programmering for CXZ8-CPU-computeren via en seriel terminal. Spectrum'en forsynes med en seriel 1.200 baud rutine mens VIC20 og CBM64 har indbygget software for dette. Sinclair maskinen sender serielt signal ud på 00 og modtager på I0 ved adresse 55. CBM-maskinerne benytter den identiske brugerport. I alle tilfælde er hastigheden 1.200 baud, 8 bit og no-parity (ingen paritet).

Tilslutninger CX81-I/O-port: Serial out: PIN 17 (bit 0 er out)
 Serial in: PIN 1 (bit 0 er in)
 Common: PIN 9

0A-0B	10-11	Pointer for GOSUB-stak. Initialiseres til 68 (HEX)
08-09	8-9	Basic-start pointer. Sættes til 1020 (HEX) fra ekstern EPROM ved autostart-op.
06-07	6-7	Pointer for toppen af GOSUB-staken. Da staken her er i registre vil R6 indeholde 00 og R7 registernumrene.
04-05	4-5	Frit register. Kan bruges til USR-rutiner
00-03	0-3	Z8671 I/O-porte

Pointerregistrene har registernumrene i den mindst betydene byte og sidenumeret 00 i det mest betydene. GOSUB-stak'en gror fra R-103 til R-58 før overflow. Når det sker vil variablerne M-Z allerede være overskrevet og dermed ødelagt. Har du flere GOSUB'en er det fornuftigst at benytte variablerne A-H. Men de kan selvfølgelig ødelægges af maskinkode. Hvis du har et simpelt program sker der ikke noget. Er du i tvivl kan du sætte den højeste variable til et kendt tal og teste om den ændres.

3.6.7 MEMORY MAP

I den følgende tabel har du en liste over hukommelsen med ekstern RAM:

ADRESSE		INDHOLD
HEX	Decimal	
.....		
FFFD	65533	BAUD-rate (hastighed). De laveste 3 bit bestemmer baud-rate som vist i afsnit 2.4.0.1.
xxFF		RAMTOP. Ekstern hukommelse for dit program.
xx00		RAMBUND.
1020	4128	Startadresse for auto-RUN EPROM-program. Gem dit program med start i byte 20.
1015	4117	Adresse for ekstern outputdriver. Her gemmes en jump-kommando til brugerens udstyr.
1012	4114	Adresse for ekstern inputdriver. Her gemmes en jump-kommando til brugerens udstyr.
100F	4111	Adresse for Interruptrequest IRQ5. Når der kommer Interrupt (afbrydelse) på IRQ5, vil Z8671 løbe gennem ROM'en til 100F og søge efter en hopadresse. Deraf navnet jump=hop. Brugerens rutiner kan så afvikles fra jumpadressen.
100C	4108	Jump for IRQ4.
1009	4105	Jump for IRQ3.
1006	4102	Jump for IRQ2.
1003	4099	Jump for IRQ1.
1000	4096	Jump for IRQ0.
0800	2048	Ekstern RAMBUND
07FF	2047	Basic/Debug ROM-top internt.
07FF-00	2047-0	Basic/Debug oversætter
00-FF	00-255	ROM - ikke adresserbar med @ eller ↑
20-27	32-39	Default konstant blok
00-0B	00-11	Interrupt request vektorer og start på Z8671 ROM

Det kan være farligt for maskinkodeprogrammerere, idet man skriver oven i hinanden uden at bemærke noget før programmet crasher. Eneste chance for at se det er når register R6-R7 krydser R8-R9. Det er derfor vi har en lille reserve til rådighed.

Hvis man ønsker at reservere dele af memory, kan man nemt ændre disse registre med LET-statements. Hvis du vil »gemme» en blok i bunden af hukommelsen kan du bruge et LET-statements til at ændre R8-R9 (%-HEX).

Brugerens program gemmes så fra den ny R8-R9 adresse. Ligeså kan du gemme program mellem GOSUB-staken og programarealet ved at gøre R10-R11 mindre.

Du kan også gemme et areal over variablerne ved at sænke R12 i multipla af 256. Det vil dog redefinere alle variable, så man skal passe på. Men man kan på den måde også skifte flere variable end A-X ind.

Hvis pointerne R8 og R10 ændres skal du NEW'e maskinen før du kan køre et program. Det vil påvirke alle andre pointere, så de stiller sig efter de ændrede. Det er MEGET vigtigt og vises bl.a. også ved vore userrutiner.

3.6.6 REGISTERSYSTEM UDEN RAM

Basic/Debug i Z8 kan faktisk godt køre helt uden RAM i meget små opgaver. Hvis den ikke kan finde nogen udvendig RAM organiserer den sig selv om så den interne stak deles med registre, liniebuffer og variable. Det begrænser selvfølgelig GOSUB-stak'en, liniebuffer-længden og uservariablerne. Uden ekstern vil nogen af de før beskrevne registre også være meningsløse.

Her har du en oversigt over Z8671 uden ekstern RAM:

ADRESSER HEX / Decimal	INDHOLD
.....	
F0-FF 240-255	Z8671 kontrolregistre - se RAM-mappe
80-EF 128-239	Her er der ingen registre til rådighed - tomt område
68-7F	Regnestak. Området gror ned fra 7F og liniebufferen gror op fra 68 (HEX).
40-67 64-103	GOSUB-stak - gror nedad
40-55 64-85	Området deles mellem variablerne M-Z og GOSUB-staken. Variablerne ødelægges af for mange GOSUB's.
22-55 34-85	Variabler fra A til Z
21 33	Fri - til f.eks. USR-rutine
20 32	Cursor position
1F 31	Interne variable - pil ikke her !
1E 30	Basic/Debug bruger dette område
1C-1D 28-29	Pointer i konstantblokken
18-1B 24-27	Interne variable - fingrene væk
16-17 22-23	Linienummer der i øjeblikket arbejdes med
14-15 20-21	Andet argument i et 3-argument USR-subrutinekald
12-13 18-19	Sidste argument og resultat u USR-subrutinekald
10-11 16-17	Basic/Debug område - kan ikke benyttes til variable
0E-0F 14-15	Næste karakter at arbejde med i input buffer
0C-0D 12-13	Linie-afslutning pointer. R12 definerer variabelsiden og holder styr på registrene fra 00.

Tilslutninger CBM-userport:

Serial out: PIN M
Serial in: PIN B plus C
Common: PIN A

Spectrum maskinen kan lave hard-copy på en Sinclair printer eller en Centronicsprinter med CX81-CEN-porten. CBM'erne må køre på de normale serielle printere eller Centronicsprintere via CD's I/O-porte til dette formål. Det er CX20-I/O eller CX64-I/O.

FUNKTIONSBESKRIVELSE

Vi har nu loadet programmet og fået menuen. Derefter vælges funktionerne:

T: RS232 TERMINAL

Skærbilledet slettes. Skærmen forbliver blank indtil Z8'eren sender noget retur. Da vi benytter fuld duplex vil en karakter fra keyboard først nå Z8'eren for derefter at blive returneret tilbage til skærmen. Skriver man et bogstav på keyboard vil bogstavet først dukke op på skærmen igen når det HAR været inde gennem Z8's hjerne. Fylder man skærmen op vil billedet scrolle.

Specielle taster:

»DELETE» omkodes af terminalprogrammet til CHR\$(8), idet denne kode af Z8 omfattes som sletning af et tegn. Når Z8 sender en CHR\$(8) til computeren, slettes den sidst skrevne karakter. »STOP» medfører returnering af MENU. (VIC20 / CBM64 = F1). »EDIT» medfører udsendelse af CHR\$(27) til Z8'eren. Det er en ESCAPE. (VIC20 / CBM64 = SHIFT Y).

G: GET Z8 PROGRAM INTO MEMORY

Henter hele Z8'eren's BASIC-program ind i udviklingscomputerens memory hvor der skal være plads til mindst 16Kbytes. Hvis der allerede ER program i memory bliver den slettet. Udviklingscomputeren sender en LIST kommando til Z8, hvorefter alle returnerede karakterer gemmes i memory. Når Z8'eren sender et »:» når listningen er slut.

P: PUT MEMORY PROGRAM TO Z8

Skriver alt hvad der er i memory'en ud til Z8'eren. Z8 vil opfatte det som om en bruger er igang med at programmere. For hvert tegn der sendes til Z8, skal udviklingscomputeren vente med at sende en ny til Z8 har sendt den netop modtagne. Det er en sædvanlig måde at sikre korrekt programoverførsel på. Når man sender en CR-karakter til Z8 skal den returnere et »:». Det er den eneste kode under 32, som sendes til Z8. Hvis Z8 ikke svarer korrekt indenfor 1 sekund meddeles »Time-out-error» i nederste linie på skærmen. Det eksisterende Z8 program slettes kun hvis linienumrene falder sammen.

H: HARDCOPY MEMORY PROGRAM

Denne funktion laver en udprintning af memoryindholdet på den tilsluttede printer. Spectrum'en finder selv ud af hvilken printer du bruger. Bruger du kommandoen UDEN printer må du BREAK og derved stoppe programmet. Ellers venter det i en uendelighed på at få forbindelse med en printer der altså ikke findes.

S: SAVE MEMORY PROGRAM ON TAPE

I nederste linie spørges »FILE NAME?». Dette navn indtastes, og memory-indholdet gemmes på bånd. Hvis du ikke indtaster noget navn, returneres til menu uden funktionen.

L: LOAD MEMORY PROGRAM FROM TAPE

I nederste linie spørges »FILE NAME?». Du skriver navnet, og loader derefter filen ind fra bånd i memory. Det gamle memoryindhold bliver overskrevet. Hvis der ikke indtastes noget filnavn returneres straks til menu.

2.4.1 VIC-20 RS232/5V TERMINALPROGRAM

Circuit Design har udviklet CXZ8-CPU-computeren til styringsformål. Det er ikke en hjemmecomputer, men et apparat man kan dedikere til forskellige opgaver så som PLC'er, Telexterminal etc.

Programmeringen køres som 1.200 baud fra en terminal. Men en terminal koster mange penge også i professionel sammenhæng. Det er årsagen til at vi hos Circuit Design syntes, at vi måtte over en anden form for programterminal. Naturligt faldt valget af terminal på de mest populære hjemmedatamater: VIC20, CBM64, Spectrum48K og de computere man ved en blanding af software eller hardware kan få til at fungere som kommercielle terminaler.

VIC20 er forholdsvis nem at have med at gøre. Den har implementeret en RS232 funktion på userporten. Funktionen kaldes via normale basicrutiner. Eneste forskel på at bruge en VIC20 som terminal er dens hastighed. Ganske vist kører den den baudhastighed men beder den om, men den holder nogle gevaldige » pauser » mellem hvert tegn. Den skal have tid til at tage data ind fra seriebufferen og føre dem displayet. Det tager LANG tid i forhold til afsendelse af et 1.200 baud tegn. Typisk 10-20 gange længere tid end at modtage selve tegnet. Derfor er faren ved at bruge en VIC20, at dens buffer bliver fyldt. Når det er sket begynder den at » glemme » data fra Z8'eren.

VIC20 kan modtage 256 karakterer uden problemer, så det er faktisk først når man beordrer Z8 til en LIST (uden argument) at det kan gå galt. Derfor vil vi nok anbefale en LIST rettet til en tilsluttet Centronicsprinter eller LIST med et argument med linienumre. Har du f.eks. 100 linier kan du faktisk nemt klare dig ved at skrive LIST 180,190 , og kun liste de 20 linier mellem de to tal. Det er nok til at fylde skærmen - men ikke VIC20's buffer.

VIC20 har sin egen norm for ASCII, og derfor vil du også se af programmet nedenfor, at vi har lavet et par statements, som ændrer CHR\$ for delete-tasten til terminalnormen. I modsat fald ville rettelser give de sjoveste resultater.

Husk i den forbindelse, at programlinierne er væk fra VIC20 når du har trykket ENTER. Vil du rette i en linie i Z8 SKAL du skrive den helt om fra VIC20-terminalen. Du kan IKKE, som på CBM-maskinerne køre rundt med cursoren i skærmen og rette det du ser. Det er jo IKKE på VIC'en det egentlige sker, men i Z8 computeren. Du kan heller ikke som på en Spectrum hive en programlinie ned med EDIT, rette og sende det rigtige resultat tilbage i maskinen. Selvfølgelig kan du hente en programlinie ud på terminalen med en LIST efterfulgt af linienummeret. Men det er kun for at se den. Rettelser skal ske ved indskrift af en ny linie.

Her har du programmet til en VIC20:

yy56-yy67
yy54-yy55
yy52-yy53
yy.....
yy22-yy21
yy20

benyttes ikke
variablen Z
variablen Y
andre variabler
variablen A
GOSUB-staken. Den gror nedad mod Basic/Debug'ens
RAM-TOP.

3.6.5 POINTER REGISTRE

Et pointer register er opbygget som 2 8-bit registre. To af disse pointere viser det øjeblikkelige indhold af liniebufferen. 4 pointere holder check på brugerens hukommelse, som vist i følgende skema:

ADRESSE (HEX)	INDHOLD (HEX)	BESKRIVELSE
ROE-OF (R14-R15)	yy68-yyF0	Næste tal i liniebufferen der skal arbejdes med. INPUT kommandoer vil reseter til begyndelsen af bufferen. IN bruger derimod alle tal i bufferen før den resetter.
ROC-OD (R12-R13)	yy68-yyF0	Sidste karakter du lægger i liniebufferen. Backspace fra terminalen trækker 1 fra denne pointer. ESCAPE vil nulstille pointeren til begyndelsen af linien. R12 er sidenummer for variable og liniebuffer.
ROA-OB (R10-R11)	yy20	RAMTOP og bunden af GOSUB-stak. Sættes ved opstart til yy20 fra højeste RAM-possition.
R06-R07	bevæges ned fra yy20	Lav det af userram'en og top af GOSUB-stak'en.
R04-R05	bevæges op fra xx00	Høj del af userram'en og stak-reserven.
R08-R09	xx00	RAM-bund. Første programlinie.

Pointerne i register R4-R5 og R6-R7 holder øje med GOSUB-stak'en og brugerens program. De gror i retning mod hinanden. Registerpointer R4-R5 angiver brugerens ramtop og den har en lille reserve, så kollision med GOSUB'en kan undgås. NEW-kommandoen sætter R8-R9 tilbage til begyndelsen af userhukommelsen sammen med stakreserven i kontrolblokken. Reserven er 32 bytes, hvilket skulle række i de fleste tilfælde.

Kun maskinkodeprogrammer vil kunne kræve mere reserve.

Pointeren R6-R7 angiver laveste GOSUB-adresse, som også er toppen af staken. Det er også en pointer for maskinkode eller interrupts, som »gror» nedad i userhukommelsen fra toppen under GOSUB'en.


```

1 REM IF THIS PROGRAM IS GOING TOO SLOW
2 REM THEN YOU HAVE TO DRIVE IN 300 BAUD3 REM PUT IN LINIE 70 .....OPEN2,2,3,
4 REM CHR$(6+32)+CHR$(32):P=0
5 REM AND SET THE Z8 TO 300 BAUD
10 REM *****
20 REM *** Z8 CBM 64 PROGRAM ***
30 REM *** COPYRIGHT BY ***
40 REM *** CIRCUIT DESIGN ***
50 REM *** DESIGN HENRIK ENIG ***
60 REM *****
70 POKE52,64:POKE56,64:CLR:OPEN2,2,3,CHR$(8+32)+CHR$(32):P=0
80 PRINT"(CLR)Z8 BASIC DEVELOPMENT SYSTEM /CD"
90 PRINT"(UP)*****"
100 PRINT"(DOWN)(DOWN)(DOWN)MENU:"
110 PRINT"(DOWN)T: RS232 TERMINAL PRESS F1 TO RETURN"
120 PRINT"(DOWN)G: GET Z8 PROGRAM INTO MEMORY"
130 PRINT"(DOWN)P: PUT MEMORY PROGRAM TO Z8"
140 PRINT"(DOWN)H: HARDCOPY MEMORY PROGRAM"
150 PRINT"(DOWN)S: SAVE MEMORY PROGRAM ON TAPE/DISK"
160 PRINT"(DOWN)L: LOAD MEMORY PROGRAM FROM TAPE/DISK"
170 GETA$
180 IFA$="T"THEN300
190 IFA$="G"THEN500
200 IFA$="P"THEN700
210 IFA$="H"THEN900
220 IFA$="S"THEN1100
230 IFA$="L"THEN1300
240 GOTO170
300 PRINT"(CLR)":POKE53272,23
310 GETB$
320 IFB$=CHR$(20)THENB$=CHR$(8)
330 IFB$="(SHIFT+Y)"THENB$=CHR$(27)
340 IFB$="(F1)"THENPOKE53272,21:GOTO80
345 GOSUB2000
350 IFB$("<)"THENPRINT#2,B$
360 GET#2,B$
370 IFB$=CHR$(8)THENB$=CHR$(20)
385 GOSUB2100
390 IFB$("<)"THENPRINTB$:
400 GOTO310
500 PRINT"(CLR) GET BASIC FROM Z8
510 PRINT"*****"
520 N=0:P=0:PRINT#2,"LIST":POKE161,0
521 GET#2,B$
522 IFB$="L"THENN=N+1
523 IFB$="I"THENN=N+2
524 IFB$="S"THENN=N+3
525 IFB$="T"THENN=N+4
526 IFN=10ANDB$=CHR$(13)THENPOKE161,0:GOTO530
527 IFPEEK(161)=1THEN521
528 GOTO600
530 I=16384:N=0
540 GET#2,B$
545 IFB$=CHR$(13)THENN=1
550 IFB$=":"ANDN=1THEN1240
552 IFB$=" "THENB$=CHR$(0)
555 B=ASC(B$)
557 GOSUB2100
558 IFB=13THENPOKEI,B:I=I+1:P=P+1:POKE161,0:GOTO580
560 IFB=32THENPOKEI,B:I=I+1:P=P+1:POKE161,0:N=0:GOTO580
570 REM
580 IFI=40960THENPRINT"(HOME)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)OUT OF ME
MORY":POKE198,0:WAIT198,1
590 IFPEEK(161)=1THEN540
600 PRINT"(HOME)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)TIME OUT ERROR !":POKE1
98,0:WAIT198,1
610 GOTO80
620 IFASC(B$)=97ANDASC(B$)=122THENB$=CHR$(ASC(B$)-32)
630 RETURN
700 PRINT"(CLR) PUT BASIC TO Z8
710 PRINT"*****"

```

Sæt så disse to programlinier ind omkring den programstump du vil måle hastigheden på:

```

100 Z = 242 : Y = @ Z
101 REM — her sætter du din programstump ind
102 Z = @ Z : S = S + AND (Y-Z-97, 255)

```

S er tiden for afviklingen af programstumpen. 97 er et offset så S ikke ændres hvis 101 udelades helt. Hvis dit program tager meget lang tid kan T1 løbe ud for tid. Den har kun 256 step. Men så kan du lægge 256 eller 512 til 97 og prøve igen. Eller hvad du nu måtte have behov for i x256.

Du kan time et program fra basic med T1 uden ekstra statements. Nemt ikke?

HUKOMMELSEN OG DENS OPBYGNING

3.6.1 HUKOMMELSESSTRUKTUR

Z8 har tre former for hukommelse,- den indbyggede ROM med Basic/Debug, de indvendige registre og plads for udvendig EPROM/RAM.

De indre registre går fra 0-255, men der er kun 144 i anvendelse. Der er 4 I/O-port-registre (R0-R3), 124 universal registre og 16 kontrol og status registre. Porte, kontrol- og statusregistre går igen i hele Z8-familien,- hvoraf vi benytter Z8671 med Basic/Debug.

Basic/Debug bruger selv en masse af universal registre til pointere, arbejdsområde og interne variable. Derfor er det farligt at benytte registre til maskinkode etc.

Basic/Debug ROM'en i Z8671 fylder 2K-bytes. Den starter på adresse 0000 og går til 2047, men da maskinen adresserer 0-256 som variabelområde, kan du kun bruge dette område med maskinkode.

Z8671 initialiseres ved opstart/reset til Basic/Debug for ekstern hukommelse.

3.6.2 INITIALISERING & OPSTART

Når der sættes strøm på Z8'eren eller du reset'er maskinen vil den først »måle» den tilsluttede RAM og derefter søge efter et automatisk opstartsprogram.

Basic/Debug tester memory fra de lave til høje adresser med mellemrum på 256 bytes. Det ville tage for lang tid at teste alle bytes, og der er næppe nogen, som finder på at montere 114-1/2-bytes ! Hvis Basic-Debug får at vide at xxFD er RAM tror den også at xx00 er det. Og det gælder faktisk også hvis der ikke er memoryfejl.

Basic/Debug sætter RAM-bunden efter den første blok den finder. Hvis den kan skrive i xxFD er xx00 også RAM. Den sætter så pointer registre R8-9 til xx00 HEX.

Basic/Debug fortsætter RAM-testen op til en position hvor den ikke kan skrive og få samme data retur. Derefter beslutter computeren sig for at RAM går til 256-bytes lavere position med yyFF som stop og yyFD som den højeste RAM-celle der kan skrives i og læses fra. RAMTOP pointeren R4-R5 sættes så til yy20.

AUTOMATISK OPSTART tillader dig at programmere din maskine til at udføre en opgave når den får strøm eller når den resettes. Autostart sker når:


```

10 FOR i=55000 TO 55730
20 PRINT i,
30 INPUT a
40 PRINT a
50 POKE i,a
60 NEXT i

```

REM : TO RETURN FROM TERMINAL MODE PRESS 'Q' AND 'SYMBOL SHIFT'

TO CLEAR THE SCREEN PRESS 'E' AND 'SYMBOL SHIFT'

55000	0	0	0	0	0	0	0	0
55008	195	235	214	195	67	216	195	166
55016	216	0	0	243	17	196	9	237
55024	83	222	214	237	91	222	214	27
55032	237	83	222	214	122	179	40	48
55040	58	233	214	61	50	233	214	40
55048	39	58	216	214	79	237	120	203
55056	71	32	224	205	227	215	254	0
55064	40	217	254	8	204	235	216	205
55072	140	217	17	196	9	237	83	222
55080	214	62	100	50	140	92	24	195
55088	17	2	0	237	83	222	214	1
55096	254	0	237	120	47	230	31	204
55104	120	215	40	175	17	196	9	237
55112	83	222	214	205	251	216	254	226
55120	40	74	254	1	40	40	254	0
55128	40	153	71	58	234	214	184	40
55136	146	120	50	234	214	205	158	215
55144	254	8	204	235	216	205	140	217
55152	62	100	50	140	92	195	243	214
55160	62	0	50	234	214	201	58	137
55168	92	71	5	205	68	14	58	136
55176	92	71	197	62	32	197	205	140
55184	217	193	16	247	193	120	50	136
55192	92	195	243	214	251	201	0	237
55200	91	220	214	205	55	216	6	8
55208	205	33	216	237	91	220	214	205
55216	55	216	31	212	33	216	220	44
55224	216	237	91	220	214	205	55	216
55232	16	240	205	44	216	58	216	214
55240	79	17	136	19	27	122	179	40
55248	110	237	120	203	71	32	245	197
55256	213	205	227	215	209	193	254	0
55264	192	24	233	237	91	220	214	203
55272	58	203	27	205	55	216	237	120

- 41 GOTO negativt eller 0 linienummer umulig.
- 44 Linienummer i GOTO-statement eksisterer ikke.
- 66 GOSUB er ikke det sidste statement i linien.
- 71 Statement kan ikke oversættes og begynder med GO...
- 81 Statement kan ikke oversættes eller "=" mangler efter LET.
- 98 LET-statement mangler sit "="
- 140 Gåseøjne " mangler i PRINT.
- 171 RETURN er ikke sidste statement i en linie.
- 172 GOSOB-stak underflow.
- 175 GOSUB for RETURN eksisterer ikke længere/for mange?
- 181 STOP er ikke i slutningen af linien.
- 207 INPUT-variabel navn mangler.
- 210 IN eller INPUT forventer variabel navn.
- 247 LIST er ikke i slutningen af en linie.
- 310 Forkert relation i et IF-statement.
- 346 Hukommelsen er fyldt op under GOSUB eller stort regneudtryk.
- 381 Deling med 0.
- 391 Manglende paranteser i AND eller USR-kald.
- 427 Syntaksfejl i regneudtryk eller uoversætteligt statement.
- 431 Manglende højre parantes i et regneudtryk.

3.4.2 INTERAKTIV DEBUG

Ofte går der lige så lang tid med at finde fejl i programmer, som at udvikle selve programmerne. Derfor kan du benytte GOTO i stedet for RUN, når et eller andet RUN'er med fejlkode til resultat. Du kan køre en linie igen og igen ved at skrive GOTO og stoppe når som helst med en ESCAPE-kode. Z8'ens UART står og venter på en ESCAPE-kode også under programafviklingen. Den vil derfor ofte være din sidste redning ud af en løkke. ESCAPE er ASCII-kode CHR\$(27).

Computeren tester for ESCAPE mellem hver programlinie. Derfor kan du ikke ESCAPE (eller BREAK) midt i en programlinie med flere statements adskilt af kolon'er. Der testes heller ikke for ESCAPE under GOTO og GOSUB, efter RETURN eller efter IF.

ESCAPE er fuldkommen sat ud af spillet hvis du har en enkelt programlinie med GOTO sig selv. Så når den ikke en ny linie med ESCAPE-test. Det samme gælder maskinkode. Her kan du også nemt lave et ubrydeligt program for dig selv - og andre.

Hvis computeren står og venter på INPUT eller IN kan du selvfølgelig heller ikke forvente, at den skal stoppe på ESCAPE-kommando. Den returnerer straks et "?" til du har fyldt input-buffere med det den beder dig om. Så kan du bryde efter dette statement og en ny linie.

Hvis du vil have maskinen til at stoppe skriver du et ord i en programlinie den IKKE kan forstå. Den bevarer så alle stakke og data og returnerer med en fejlkode. Stopper du den i stedet med STOP-kommandoen under interaktiv debugning, vil du cleare GOSUB-staken.

RUN-kommandoen er altid direkte. Den bruges til at starte programafviklingen. Data for første IN-kommando kan indtastes SAMMEN med RUN adskilt af komma mellem RUN og den afsluttende CR (ENTER).
Det kan se sådan ud:

RUN 45, -583

3.3.19 STOP KOMMANDO

SYNTAX

stop-stmnt => STOP
EKSEMPEL
STOP

STOP bruges i enden af et program der er slut. STOP kan sættes ind hvor som helst i et program, så man hen ad vejen kan teste hvordan programafviklingen forløber. STOP-kommandoen vil altid placeres automatisk efter programmet. Derfor kan du helt undvære denne kommando når dit program fungerer. Det sparer hukommelse.
Programafvikling kan køres stepvis til STOP-kommandoer med en GOTO-kommando. Hvis f.eks. linie 140 til 160 skal prøves sætter man en STOP-kommando i 165 og skriver GOTO140. Så kører det een gang gennem 140 til 165 og returnerer programmets resultat.

FEJLKODER OG INTERAKTIV DEBUG

3.4.1 ERROR/FEJL

Hvis du skriver noget forkert i det Basic-sprog Z8'eren nu engang er skabt til at forstå, må den selvfølgelig give op. Den kan ikke oversætte ord den ikke har lært. Derfor vil den returnere en fejlkode med angivelse således:

(fejlkode NR.) AT (linie NR:)

Når computeren har opdaget fejlen ved forsøg på programafvikling hopper den over i direkte programmeringsmode. Så kan man rette eller skrive en ny linie. Hvis du prøver at skrive en kommando i direkte mode maskinen ikke er i stand til at oversætte, vil den returnere en klokkekode til en terminal. Bell, som koden hedder har ASCII-tallet 7.

Her har du en liste over de fejl Z8 kan opdage og giver dig fejlkode for:

- 0 Programmet afbrudt af ESCAPE-kode input eller flag-bit = 0.
- 11 Programmet har fået et for stort linienummer over 32768 el. 0.
- 17 Hukommelsen er fyldt, - sidste linie er ikke modtaget.
- 26 Intet program i maskinen til at RUN'e.
- 37 GOTO er ikke sidste statement i en programlinie.

55280	230	1	203	71	192	6	8	197
55288	237	91	220	214	205	55	216	237
55296	64	203	24	193	31	16	240	237
55304	91	220	214	205	55	216	237	64
55312	203	191	254	10	32	2	62	0
55320	79	120	238	1	230	1	71	121
55328	201	245	58	216	214	79	62	0
55336	237	121	241	201	245	58	216	214
55344	79	62	255	237	121	241	201	245
55352	27	122	179	32	251	241	201	1
55360	255	255	201	205	204	216	243	17
55368	0	64	235	167	237	90	34	217
55376	214	235	62	76	205	158	215	254
55384	76	32	70	62	73	205	158	215
55392	254	73	32	61	62	83	205	158
55400	215	254	83	32	52	62	84	205
55408	158	215	254	84	32	43	62	13
55416	205	158	215	254	13	32	34	205
55424	197	215	254	0	40	27	119	35
55432	237	91	217	214	235	167	237	82
55440	40	10	235	254	58	32	232	1
55448	0	0	251	201	1	1	0	251
55456	201	1	2	0	251	201	205	204
55464	216	62	1	185	200	243	1	0
55472	0	126	254	58	40	17	205	158
55480	215	254	58	204	197	215	190	35
55488	24	236	1	1	0	251	201	251
55496	1	0	0	201	42	75	92	62
55504	208	190	40	11	62	128	190	40
55512	14	205	184	25	235	24	240	17
55520	6	0	25	1	0	0	201	1
55528	1	0	201	62	8	205	140	217
55536	62	32	201	33	40	0	201	33
55544	80	0	201	205	142	2	33	0
55552	0	122	254	39	204	243	216	254
55560	24	204	247	216	22	0	25	17
55568	21	217	25	126	201	66	72	89
55576	54	53	84	71	86	78	74	85
55584	55	52	82	70	67	77	75	73
55592	56	51	69	68	88	0	76	79
55600	57	50	87	83	90	32	13	80
55608	48	49	81	65	0	98	104	121
55616	54	53	116	103	118	110	106	117
55624	55	52	114	102	99	109	107	105
55632	56	51	101	100	120	0	108	111
55640	57	50	119	115	122	32	13	112
55648	8	49	113	97	0	42	94	0
55656	38	37	62	0	47	44	45	0
55664	39	36	60	0	63	46	43	0
55672	40	35	1	0	96	0	61	59
55680	41	64	0	0	58	27	13	34
55688	95	33	226	226	71	58	137	92
55696	254	4	48	22	58	136	92	254
55704	1	40	5	120	254	13	32	10
55712	62	24	50	137	92	62	33	50
55720	136	92	120	215	201	215	201	0
55728	0	0	0	0	0	0	0	0


```

50 LET ORG =55000
51 LET PORT =7
52 LET HASTIGHED =110
60 POKE ORG, PORT
61 POKE ORG+4, HASTIGHED
100 GO TO 8000
110 CLEAR 55000
120 LOAD "ps-z8.code" CODE 55000,730
130 GO TO 10
1000 REM // RS232 TERMINAL //
1002 CLS
1006 PRINT "begin :": PRINT
1010 RANDOMIZE USR (ORG+8)
1040 RETURN
2000 REM // PRG TO MEMORY //
2005 CLS
2006 PRINT "LIST DUMP"
2010 DIM P$(1484)
2020 LET ERR= USR (ORG+11)
2030 IF ERR (<) 0 THEN GO SUB ERRORLIST
2040 RETURN
3000 REM // MEMORY TO Z8 //
3005 CLS
3010 IF I= USR (ORG+14) THEN PRINT "ERROR (no connection ?)"
3015 PRINT ,,,,"press any key"
3020 PAUSE 0
3030 RETURN
4000 REM // HARDCOPY //
4010 LET I=1
4020 IF P$(I)=":" THEN RETURN
4030 LPRINT P$(I);
4040 LET I=I+1
4050 GO TO 4020
5000 REM // SAVE PRG //
5010 INPUT "FILE NAME :",F$
5020 IF F$="" THEN RETURN
5030 SAVE F$ DATA P$()
5040 RETURN
5500 INPUT "FILE NAME :",F$
5510 IF F$(1)=":" THEN RETURN
5520 LOAD F$ DATA P$()
5530 RETURN
8000 REM RANDOMIZE USR INIT
8010 LET MPIL=1
8100 CLS
8110 PRINT AT 5,0;"MENU:"
8120 PRINT
8130 PRINT "RS232 TERMINAL", TAB 30;"T",,"GET Z8 PROGRAM INTO MEMORY
EMORY PROGRAM TO Z8 P",,"HARDCOPY MEMORY PROGRAM H"
8140 PRINT
8150 PRINT "SAVE MEMORY PROGRAM ON TAPE S LOAD MEMORY PROGRAM FROM TAPE
8160 LET S$= INKEY$
8170 IF S$="" THEN GO TO 8160
8180 IF CODE S$>90 THEN LET S$= CHR$ ( CODE S$-32)
8190 IF S$="T" THEN GO SUB 1000
8200 IF S$="G" THEN GO SUB 2000
8210 IF S$="P" THEN GO SUB 3000
8220 IF S$="H" THEN GO SUB 4000
8230 IF S$="S" THEN GO SUB 5000
8240 IF S$="L" THEN GO SUB 5500
8250 GO TO 8100
9000 CLEAR 30000
9001 PRINT "SAVE PROGRAM"
9002 SAVE "PS-Z8" LINE 110
9005 SAVE "ps-z8.code" CODE 55000,730

```

trokarakter. Derfor vil Basic/Debug's interne cursor ikke altid stå det rette sted hvor der printes - i dette tilfælde. Her vil der være fare for at et komma-skilletegn kan kikke:

```

10 X=0
20 PRINT "X'G",X
30 PRINT "X",X

```

Køres det program får du på terminalen:

```

X 0
X 0

```

Statement i linie 20 indsætter nemlig kun 7 mellemrum fordi control'G skubbes 1 plads til højre for cursorens aktuelle plads.

3.3.16 REM KOMMANDO SYNTAX

```

rem-stmnt => REM (alle følgende karakterer med undtagelse af CR)
EKSEMPLER
REM KONTROL LOOP
REM SUBROUTINE NAVN
REM KODE FORKLARING

```

REM-kommandoen er en label du klistrer på dit program, hvor en forklaring på funktionen er ønskværdig. Du kan skrive hvad som helst uden at det har indflydelse på programafviklingen. Det fylder bare godt op i memory'en. Så spar REM'erne hvis du mangler plads.

3.3.17 RETURN KOMMANDO

SYNTAX

```

return-stmnt => RETURN | RET
EKSEMPEL
RETURN
RET

```

RETURN skal altid efterfølge den sidste kommando i en subrutine. Ellers kan den ikke finde tilbage i programmet. Forkortelsen RET er lige så god og behandles på fuldkommen samme måde.

En RETURN-kommando fører tilbage til den nærmeste anden RETURN-kommando i rækkefølge på GOSUB-staken. Derfor kan GOSUB'er med RETURN lægges inden i hinanden som kinesiske æsker.

3.3.18 RUN KOMMANDO

SYNTAX

```

run-stmnt => RUN [ udtryk ( ' , ' udtryk ) * ]
EKSEMPLER
RUN
RUN 17, %200, 23

```


Når man skriver en PRINT-kommando vil Basic/Debug liste decimalværdierne på terminalen. Kun det betydende ciffer udskrives; nuller foran fjernes og tal behandles som heltal med fortegn. Senere viser vi en metode til udskrift af tal uden fortegn.

Hvis du ønsker at printe en HEX-værdi ud bruger du følgende syntax:

PRINT HEX (udtryk)

Basic/Debug udregner udtrykket og skriver dens positive hexadecimale ækvi-valent ud på terminalen. PRINT-kommandoer kan ikke udskrive negative hexadecimale tal.

Til forskel fra karakterstrengene skal HEX-tal ALTID have en PRINT-kommando foran. Undtagen dog hvis man giver tallet et plus eller minus fortegn.

Så går det alligevel.

Bruger man et komma mellem hvert PRINT-statement skilles de i udskriften som i en TAB-kommando. Hver tab er på 8-pladser i søjler over hele skærmen. Hvis du vil udprinte søjler i kolonner skal du blot adskille dine PRINT-kommandoer med komma'er i lange rækker. Hvis dine udprintninger i hver statement er længere end 8 karakterer hopper TAB'en til næst efterfølgende TAB.

Hvis du vil have printet ting ud efter hinanden uden mellemrum på samme linie bruges skilletegnet ; Hvis du ikke bruger noget tegn vil hver PRINT-kommando komme på sin egen linie. De mest brugte PRINT-statements ser således ud:

PRINT "OUTPUT=" ; X

Det vil udskrive en følgetekst med et variabeltal direkte efter. Hvis du afslutter PRINT sætninger med et semicolon, sker der ikke noget lineskift efter udprintningen.

Ønsker man at udskrive højreudsluttede tegn og tekster må man altid skrive PRINT-kommandoer med gæseøjne uden om space-tegnet således:

```
200 IF N < 10000 THEN PRINT " ";
210 IF N < 1000 THEN PRINT " ";
220 IF N < 100 THEN PRINT " ";
230 IF N < 10 THEN PRINT " ";
240 PRINT N
```

Basic/Debug kan udskrive de fleste kontrolkarakterer så som klokke (bell) hvis du bruger gæseøjne. Det gælder dog ikke følgende karakterer:

Back-Space	('H)
Escape/Cansel	(ESC)
Carriage-Return	(CR)
Linefeed	(LF)
Slet/Delete	(DEL)
Nul/Null	(NUL)

Apostrofen » ' » foran H angiver at kontroltasten skal holdes nede hvor denne findes. Større RS232-terminaler vil ikke flytte cursor når du bruger en kon-

KOMPONENTLISTE

CXZ8-CPU

Z8 computeren samles med sokler meget omhyggeligt. Det er bedst at lodde et komponentben i hver ende og klippe benene af HELT tæt til printet. Det er nemlig af typen: Dobbeltside gennempletteret. Derfor passerer alle ledere gennem hullerne - som altså IKKE må bores større. Tinet suges op i hullet så lodningen bliver helt flad. Men tilføj ikke for meget tin eller varme. Det kan danne loddetinklumper på printets komponentside - katastrofalt og umuligt at opdage hvis tinnet suger sig ind under en IC-sokkel. Så pas godt på. Benyt sokler til alle IC'erne, så udskiftning er mulig ved service. Vedr. garanti: Circuit Design yder garanti på printpladen. Den ombyttes til en ny hvis den ubenyttede printplade har fabrikationsfejl.

Men Circuit Design yder IKKE garanti på dit eget arbejde eller det færdige produkt - ej heller hvis det er købt som samlet/kit i Medlems-Service. Hvis HELE produktet er købt i kit eller samlet i medlemsservice, kan service eller afprøvning foregå til sædvanlig timebetaling mod forudgående aftale.

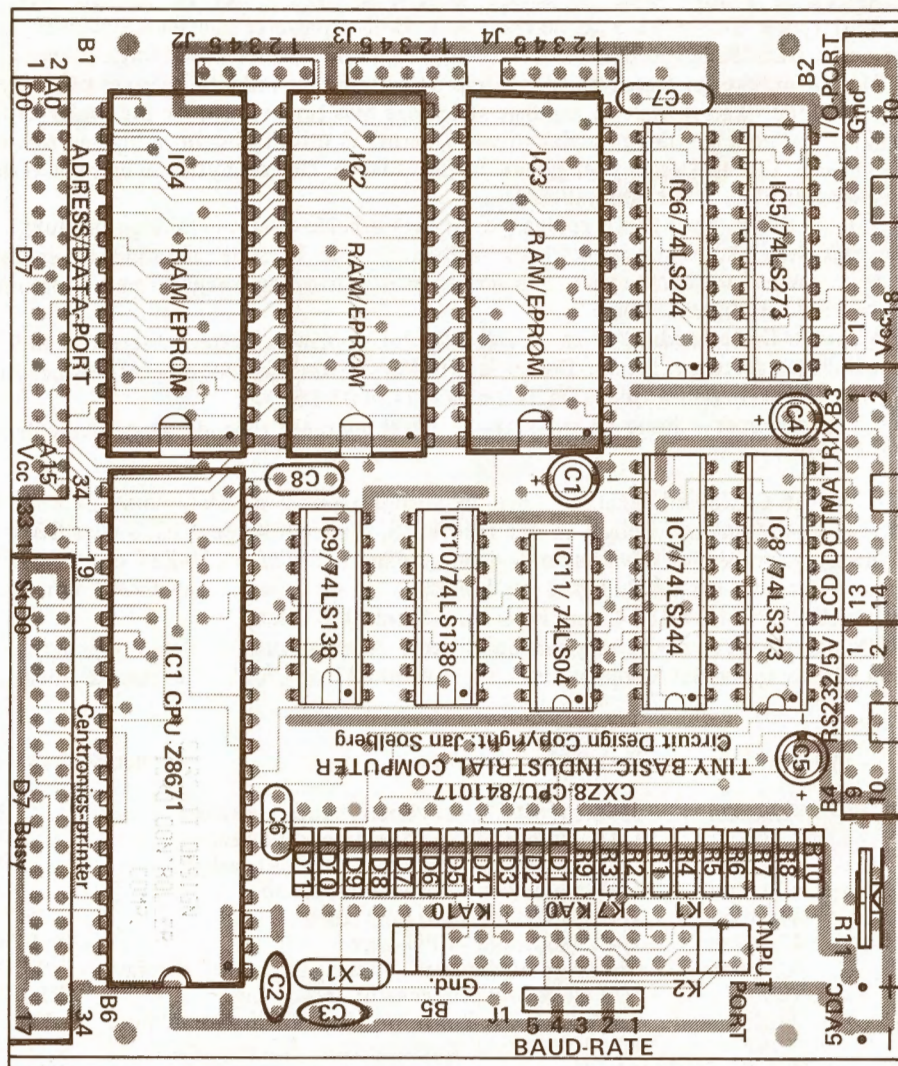
Men bemærk: Dette produkt er på et højt teknologisk plan og sætter særlige krav til brugerens indsigt i software. Circuit Design kan IKKE yde undervisningspræget service, - kun normal telefonservice hver fredag fra 14-16 på telefon 03 146046.

Diverse programmer i Basic og maskinkode fremkommer løbende gennem medlemsbladet Circuit.

Til brugen af CXZ8-CPU skal du have en strømforsyning på 5V/300mA min. og en computer eller terminal. Billigst er det med en Spectrum48K, en VIC20 eller CBM64. Terminalprogrammer for udvikling over disse maskiner: PS-Z8T, PC20-Z8T og PC64-Z8T. Ønsker du at gøre dit udviklingsudstyr komplet, må du også have en EPROM-brænder af typen CX81-PRM/CX81-64, som kører direkte fra Z8-printet!

EPROM-brænderen er nødvendig hvis du vil lagre de brugerrutiner denne vejledning beskriver. De komponenter Medlems-Service leverer til KIT og SAMLET er mærket med »:

Nr.	incl. Værdi	Beskrivelse - funktion	Medlems-Service nr:
R1	» 4,7kohm	1/4W modstand - keyboard/baud-rate select	I4K7
R2	» 4,7kohm	1/4W modstand - keyboard/baud-rate select	I4K7
R3	» 4,7kohm	1/4W modstand - keyboard/baud-rate select	I4K7
R4-8	» 4,7kohm	1/4W modstande - keyboard pull-up	I4K7
R9	» 470 ohm	1/4W modstand - I/O-port reset	I470E
R10	» 470 ohm	1/4W modstand - CPU-reset	I470E
R11	» 4,7 kohm	trimmepotentiometer - viewing angle LCD-display	JT4K7
C1	» 100uF	10V elektrolytkondensator - afkobling	KE100U
C2-3	» 27pF	100V keramisk kondensator - CPU-X-taloscillator	KK27E
C4	» 4,7uF	40-63V elektrolytkondensator - I/O-port reset	KE004U
C5	» 4,7uF	40-63V elektrolytkondensator - CPU-reset	KE004U
C6	» 47nF	50V polyesterkondensator - afkobling	KP047K
C7	» 47nF	50V polyesterkondensator - afkobling	KP047K
D1-10	» 1N4148	50V/100mA siliciumdiode - keyboardselect	H-1N4148
IC1	» Z8671	40-pin sokkel med ZILOG CPU/Basic-Interpreter	HZ8671
IC2	» 2716-64	28-pin s. 2-8K EPROM med USR-rutiner anbragt nærm.J3	-



3.3.12 NEW KOMMANDO

SYNTAX

new-stmnt => NEW

EKSEMPEL

NEW

Skriver du NEW til Z8'eren vil pointeren R10-11 blive resat til start af user-hukommelsen. Resten af hukommelsen betragtes som tom og kan da overskrives med nyt program.

Hvis du kommer til at NEW'e ved en fejl er dit program faktisk IKKE smadret. Du kan under alle omstændigheder liste det ved at sætte linienummeret for første linie på et lavt tal:

LET $\uparrow\uparrow 8 = 1$

Selvom det ser ud som om du kan få det NEW'ede program til at køre, er det lidt farligt. For du vil ikke længere have sikkerhed for memory overflow. Og DET kan ødelægge dit program fuldkommen.

3.3.13 PRINT KOMMANDOEN

SYNTAX

```

print—stmnrt => PRINT [ tegn (skilletegn) ★ ]
skilletegn   => ', ' | ', ' ;
tegn         => afmålt—streng | udtryk | HEX '(udtryk)'
bestemt—streng => afmålt—streng | fortegn—udtryk | '(udtryk)'
afmålt—streng  => ' ' ' ' (enhver karaktersekvens uden nuller,
                    slettekarakterer, LF/CR-enter, escape,
                    backspacer eller gåseøjne ' ' ' ' )

```

EKSEMPLER

```
PRINT HEX (225)
"SVARETER";X
(A★100)
+%800+Z
PRINT A,B,C,D,E
```

PRINT-kommandoen LIST'er sit eget argument, som kan bestå af tekst, tegn eller tal til den tilsluttede terminal. Skilletegne har betydning for hvordan du får udskrevet meddelelsen fra PRINT.

Alle karakterer og mellemrum i en PRINT-sætning skrives ud som de er skrevet ind. Dog med undtagelse af gåseøjnene ", som netop er PRINT statement'ets kendetegn. Hvis du VIL have gåseøjnene med i en tekst må du sætte et par uskyldige apostrofer i stedet.

Kendetegnet for en PRINT kommando er netop gåseøjnene", som kan benyttes alene. Det sparer memory. PRINT-kommandoer alene giver en tom linie i udskriften. PRINT-kommandoer kan følges med andre kommandoer i en programlinje hvis du bruger skilletegnet ':':

I den kommando udføres først en memory-beregning, derefter beregnes udtrykket og endelig sættes det beregnede tal ind i hukommelsen. Da der er tale om et 16-bit ord bliver den mest betydende byte lagret først i adressecellen som skrevet. Den efterfølgende mindst betydende byte a' 8 bit lagres i cellen lige efter. Pas derfor godt på når du roder rundt i de indre registre med program. Det kan få katastrofale følger for dit program - CRASH !!!

3.3.10 LIST KOMMANDO

SYNTAX

list-stmnt => LIST [startlinie—linie [' , ' slut—linie]]

start-linie => udtryk

slut-linie => udtryk

EKSEMPEL

LIST

LIST 200,1000

Kommandoen bruges interaktivt til når som helst at skrive et program ud på skærm eller printer. LIST vil give dig hele programmet.

Bruger du i stedet LIST efterfulgt af et enkelt linienummer, vil kun den angivne linie blive listet. To linienumre efter kommandoen vil liste fra linien med det første til det sidste tal. Komma bruges som skille tegn.

LIST-kommandoen kan også bruges i programmer som »tekstbehandling«. Det er fordi Basic/Debug ikke tjekker programmet for det skal eksekveres.

Du kan narre computeren til at behandle tekst således:

```

100 REM DETTE PROGRAM SKRIVER EN TEKST N GANGE
110 IF N > 0 THEN 200
120 : PRINT "HVOR MANGE GANGE ?";
130 : INPUT N
200 REM LOOPET BEGYNDER HER
210 : LET N = N-1
220 : LIST 100,1070
230 : IF N > 0 THEN 210
240 STOP

1000 | FØLGENDE TEKST ER GEMT I MASKINENS
1010 | HUKOMMELSE. TEKSTEN VIL BLIVE UDSKREVET NÅR
1020 | PROGRAMMET RUN'ES. HVIS DU PRØVER AT EDITERE
1030 | ELLER RUN'E LINIE 1000 TIL 1050 FÅR DU EN FEJL-
1040 | MEDDELELSE. MEN I PROGRAMMET OVENFOR VIL
1050 | DET IKKE BLIVE EKSEKVERET, KUN LISTET.
```

Fem af linierne i dette program viser blot programopbygningen. Dem kan du helt udelade i dit rigtige program. Kolon foran kommandoerne sikrer blot at computeren ikke fjerner mellemrummene, som giver en strukturel opbygning. Teksten bliver udskrevet nøjagtig som de ser ud på linierne 1000 til 1050. (Incl. linienumre).

Stregerne til venstre i teksten kan benyttes til at skabe indrykninger i teksten. Udelader du dem bliver teksten venstrestrekket fordi mellemrum i starten fjernes.

IC3	»	2716-64	28-pin s. 2-8K EPROM/RAM anbragt nærmest J4	-
IC4	»	6116	28-pin sok.m. 2K-RAM (evt.5117) anbragt nærm. J2	H-M6116
IC5	»	74LS273	20-pin sokkel m.udg. f.bl.a. EPROM-brænder	H74LS273
IC6	»	74LS244	20-pin sokkel m.indg. f.bl.a. EPROM-brænder	H74LS244
IC7	»	74LS244	20-pin sokkel m.indg. f. keyboard input	H74LS244
IC8	»	74LS373	20-pin sokkel m.udg. f. adress-decoding	H74LS373
IC9	»	74LS138	16-pin sokkel for adress decoding	H74LS138
IC10	»	74LS138	16-pin sokkel for adress decoding	H74LS138
IC11	»	74LS04	14-pin sokkel for hex-inverter	H74LS04
B1	»	34-pol	34-pol pcb data/adress/control connector	DDIL3402
B2	»	20-pol	20-pol vinkel fladkabelkonnektor for I/O-port	DDIL2000
B3	»	14-pol	14-pol vinkel fladkabelkonnektor for display	DDIL1400
-	-	-	14-pol kabel for display: DDIL1421	-
B4	»	10-pol	10-pol vinkel fladkabelkonnektor for RS232/5V	DDIL1000
-	-	-	10-pol kabel for RS232 til computer: DDIL1010	-
B5	»	20-pol	20-pol keyboard fladkabel konnektor	DDIL20PC
B6	»	34/36-pol	36-pol centronics konnektorkabel til printer	DCENTRON
X1	»	7.3728MHz	CPU krystal - nøjagtigt af hensyn til baudrate gen.	SX-73728

Desuden benyttes: 2 loddeøjne type AL, printplade CXZ8-CPU, div. jumperlus ved J1-4 og en vejledning CXZ8-CPV.

Printplade og beskrivelse kan leveres separat hver for sig men medfølger kit'et: CXZ8-CPK og den samlede enhed CXZ8-CPS. Printet er tilpasset en ALU-box af typen B2010-B2015 eller kan skæres til for europakort rack's. 5V akkumulator med 4 NiCd batterier på 500mA/H (CD: O-PENA) i en 4-rum's batteriboc (CD: F404) eller 3 batterier på hver 1,5V (CD:O-BABY) og en 3-rum's batteribox (CD: F403). Batterilås F401.

Evt. kan strømforsyning bygges separat i boxen T2605 med T2605 transformator og 5V-forsyningsstabilisator (CD: CM26-05 lev. januar-85).

CPU'en vil senere blive udvidet med I/O-modul, RAM-modul, AD/DA-modul, stepper-motorstyring mv.

3.1.1 DESIGN FORUDSÆTNINGER

Det BASIC-sprog vi forudsætter du kender fra f.eks. Sinclair, Commodore eller Regnecentralen, benyttes faktisk også i CXZ8-maskinen. Men til forskel fra de store maskiner og moderne hjemmecomputere, » taler » Z8'eren et mere beskedent BASIC-sprog. Der mangler trigonometriske funktioner, matematiske funktioner, array og streng-handling. Desuden kan Z8'eren kun klare heltal fra minus 32.000 til plus 32.000 og alle redundante kommandoer er fjernet. Redundante kommandoer er kommandoer man kan sammensætte eller udføre med grundkommandoerne.

Det er gjort for at spare plads i hukommelsen og for at få computeren til at arbejde hurtigt - Z8'eren ER altså en » opgavecomputer » - ikke en hjemmecomputer. Men hvis du VIL udvikle den til en hjemmecomputer kan det godt lade sig gøre. Så er du bare bundet til DEN opgave. Eller mangler du en ultra smart PLC-styring? Så er CXZ8 perfekt. Hurtig, nem at programmere selv » i marken » og billig i forhold til andre industristyringer.

Men læg samtidig mærke til at vi har udbygget selve grundfunktionen med et antal porte for I/O, keyboard, printer etc. MERE end hvad mange hjemmecomputere har i hardware ved leveringen! Men det er gjort for at sikre at en masse mindre opgaver kan løses UDEN ekstra indkøb af apparater og moduler.

Den dialekt Z8'eren taler - Basic/Debug - er forenklet i forhold til hjemmecomputere. Men den tillader meget hurtig test af hardware, software og hurtige modifikationer på enhver memoryadresse, bit for bit test af enhver port, bit-manipulationer og logiske operationer. Desuden kan Basic/debug arbejde med decimaltal OG HEX-tal direkte på input og output, ligesom man kan indføje maskinkodekommandoer eller subrutiner efter eget ønske hvor som helst. Det har vi bl.a. gjort i den ekstra EPROM, som har keyboard-scan og printerudgang. Der er IKKE normalt for Basic/Debug i grundversionen.

Men mest interessant er det at Basic/Debug ved power-up og reset starter DIN programafvikling i stedet for at gå i stå, som en hjemmecomputer ville gøre det. Og CXZ8-CPU'printet har ud over portene også plads til dine programmer i RAM og EPROM.

3.1.2 BASIC/DEBUG PROGRAMAFVIKLING

Basic/Debug kan eksekvere dine ordrer på to måder. Enten over et program ved efterfølgende RUN eller direkte som enkeltordrer. Systemet er klar til at acceptere en kommando når det har sendt dig et kolon :

Hvis du vil udføre en direkte kommando skriver du f.eks. blot PRINT. Den kommando udføres når du har trykket CR (ENTER). Det er CHR\$(13) i ASCII-alfabetet.

Nogle af kommandoerne har kun mening hvis de eksekveres direkte. Det gælder således RUN, LIST og NEW.

Andre så som GOTO og LET kan benyttes både direkte og i programmer. Når du har skrevet RUN vil Z8'eren afvikle det program, den har i sin hukommelse og derefter returnere til kommando-mode igen. Det kan dog også ske ved en ERROR (fejl).

Hvis kommandolinien slutter af med en variabel dukker »?» op på terminalen. Men generelt er det nemmest at bruge en LET-kommando til at assigne en variabel til et tal i direkte mode.

For at hjælpe operatøren ved terminalen vil man normalt indlede en IN eller INPUT kommando med en PRINT-kommando, så operatøren ved hvad der bedes om. Indleder du med PRINT ; vil " ? " fra INPUT komme i samme linie som PRINT skrev i.

Selv om Basic/Debug ikke har strengbehandling, kan man godt bruge en enkelt inputkarakter til svarresponse:

```
100 PRINT "SVAR JA ELLER NEJ"
110 LET N=J-1
120 PRINT "FIK DU DET ?"
130 INPUT N
140 IF N=J THEN PRINT "FINT !"
```

I dette eksempel betyder J ikke noget. Operatøren kan skrive JA, JAN, JUBII-JAY, da variabelen N er lig J.

Skriver du derimod N, NEJ eller NIX-BIX, forbliver variabelen N uforandret og er forskellig fra J. Ønsker du at tjekke andre bogstaver end J eller N, kan du bruge en helt forskellig værdi for J, f.eks. -32323 og tjekke både J og J + 1 efter input'et.

3.3.9 LET KOMMANDO

SYNTAX

let-stmnt => [LET] venstre - del '=' udtryk
venstre - del => variabel | '@' faktor | '↑' faktor

EKSEMPLER

```
LET A = A + 1
@ 1020 = 100
↑ 8 = % 100 ★ C
```

LET tildeler udtrykket en talværdi eller memoryplads. Venstre del af udtrykket kan være et bogstav mellem A og Z, en memory-plads eller en registerplads. Udtrykkets talværdi gemmes så enten på variabelens plads eller på den angivne memory plads. Lighedstegnet er helt specielt for denne syntax, så derfor kan keyword'et LET altid udelades. Desuden kan en variabel tildeles en værdi, som afhænger af variabelen selv.

```
LET A = A + 1
eller,
A = A + 1
```

LET-kommandoen kan også bruges til at gemme værdier i memory'en. Det fungerer som POKE-kommandoer i andre Basic-dialekter:

```
LET @ 1024 = B / 2
```


IF Z > % 1000 THEN A = Z

Uanset hvor mange mellemrum du ville placere mellem det hexadecimale tal og variabelen ville det blive forkert uden THEN. For inden i maskinen udelades mellemrummene.

3.3.7 INPUT/IN

SYNTAX

input-stmnt => INPUT variabel (',') variabel) ★
in-stmnt => IN variabel (',') variabel) ★

EKSEMPLER

IN C, E, G

INPUT A

Disse statements kræver svar fra operatøren efter at have returneret et »?» til terminalen. Så hentes tallet ind og gemmes i en indiceret variabel. Flere skilles med komma. Hvis du ikke skriver et antal input's som svarer til programmets ønske, vil terminalen modtage »?» indtil du har dem alle inde.

De to kommandoer er forskellige i den form de henter data.

Maskinen accepterer også generelle udtryk i en input-kommando. Har man således givet en variabel en værdi et sted tidligere i programmet, kan input godt ændre variabelen. INPUT C, A kan f.eks. klare: 10, C ★ 5. INPUT vil nulstille input bufferen og afvente indtastning mens IN vil anvende evt. resterende værdier i bufferen før operatører behøver at taste noget ind.

I programmer hvor man skal indskrive flere input's i række, kan de skilles med komma'er. Kommaerne kan udelades hvis de ikke indgår i Basic/Debug oversættelsen direkte. Mellemrum ignoreres altid.

Følgende eksempel viser hvordan variabler kan blive oversat eller ikke oversat i Basic/Debug:

? % 1 2 3 ,A, ND (56)	(hex 123, variabler A, N, D, decimal 56)
? % 12 3AND(56)	(hex 123A, variabler N, D, decimal 56)
? % 1 2 3, AND (56)	(hex 123, værdi 56 AND'es med sig selv)

Da Basic/Debug kun har en liniebuffer, vil INPUT og IN blive behandlet forskelligt i programmer og direkte kommandomode.

I direkte kommandomode vil terminalen for hvert input overskrive det samme i bufferen igen og igen, og dermed hele tiden slette de foregående data. Derfor vil variable efter INPUT kun assigne første karakter som data. Assigne betyder af forbinde med..

Men kommandoen IN kan assigne hele lister af variabler til udtryk i direkte mode:

IN A, 10, B, 15, C, 20

Basic/Debug snupper den første variabel og hopper til næste i keyboardbufferen. INPUT ville sende besked til terminalen med et »?» for hvert tal mens IN bare bruger alle tal i bufferen før den så sender »?».

Derefter vil IN assigne tallet 10 til A, 15 til B osv. til alle tal er brugt.

3.1.3 SYNTAX

Programmet er en serie af instruktioner, som lagres efter hinanden i maskinens RAM med en linie ad gangen. Computeren læser så instruktionerne fra venstre mod højre. Programlinien skal bestå af et linienummer og en kommando som:

100 PRINT "CIRCUIT DESIGN"

Linienummeret fortæller maskinen, at den ikke skal udføre kommandoen straks, men vente til man skriver RUN. Derfor gemmes den i maskinens programhukommelse. Hvis man har skrevet andre programlinier med andre linienumre gemmes de også i hukommelsen i nummerorden fra 1 til 32.767.

Når man skriver programlinien benytter man normalt et mellemrum mellem linienummer (100) og kommando (PRINT). Glemmer man det indsætter computeren selv det manglende mellemrum (space) og sætter man for mange ændrer maskinen det til kun eet. Men i hukommelsen spares mellemrummet. Derfor er det kun via terminalen man ser mellemrummene. F.eks. via LIST. Hvis man vil benytte mange kommandoer kan man snyde for linienumre ved at skille med tegnet kolon »:». Det sparer plads, men kan også blive ganske uoverskueligt. Eneste restriktion er at en linie ikke må indeholde mere end maksimalt 130 tegn.

Basic/Debug i Z8 ignorerer store og små bogstaver fra terminalen. Skriver man store og små blandet vil alle karakterer blive oversat til store (UPPER CASE).

Normalt har kommandoen i et program også et ARGUMENT. Ovenfor er kommandoen PRINT efterfulgt af argumentet CIRCUIT DESIGN. Men den indbyggede debug'er kan også klare kommando-ord og argumenter i forkortet eller helt manglende form. Her er f.eks. et par muligheder:

PRINT

IF C () USR(A) %500

@ % 1020 = 100

"SVARET ER"; X ; "BANANER"

Basic/Debug kan oversætte 15 forskellige »keyword's», dvs. stavede kommando-ord. Hvert keyword specificerer et såkaldt »statements» - en ordre til at udføre en enkelt programinstruktion. Statement's kan udføre 3 ting:

Give en variabel en værdi (LET). Det kaldes at »assigne». Man siger f.eks. at LET A = 23 eller LET A = B er at assigne A til værdien 23 eller f.eks. B.

Andre input- og output statements er f.eks. INPUT, IN eller PRINT.

Endelig er der en række »flow» statements, som benyttes i programafviklingen som: IF, GOTO, GOSUB, RETURN og GOA.

Argumentet kan også benyttes alene. Men mest i direkte kommandoer:

(4096)

A x B x C

@ % 1020

Vore grafiske muligheder er desværre lidt begrænsede her. Bogstavet lille »a» betegner således specialtegnet »snabel-a» på terminalen.

3.1.4 BASIC/DEBUG EDITOR

Bsaic/Debug i Z8 har indbygget linie-editor. Derfor vil maskinen undersøge alle programlinier før de »godkendes» og indlægges i hukommelsen.

Hvis du vil se dit program skriver du blot LIST (og CR/ENTER), hvorefter det i hukommelsen indeholdte program udskrives på skærmen.

Basic/Debug lægger linierne ind i nummerorden. Derfor vil du ikke kunne skrive programlinier med samme nummer. Skriver du en linie med et nummer der allerede findes vil den gamle linie blive slettet og den nye indlagt. Skriver du blot et linienummer efterfulgt af CR, vil enhver gammel linie med samme nummer blive slettet.

Hvis du opdager en fejl i en linie (eller maskinen giver fejl), kan du backspace dig gennem linien til fejlen og skrive det rigtige. Backspace sletter det du kører over i liniebufferen. Backspace er CHR\$(8) fra terminalen.

Ofte er backspace det samme som DEL eller DELETE.

Skriver du flere backspace end der er karakterer i en linie bliver den slettet. Vil du have slettet liniebufferens indhold kan du også skrive ESCAPE karakter.

BASIC/DEBUG UDTRYK

3.2.1 INTRODUKTION

Et udtryk i Basic/Debug repræsenterer en numerisk værdi, som benyttes til programudførelsen. Det kan være opbygget af følgende Basic-elementer:

- 1) Konstanter
- 2) Variable
- 3) Operander
- 4) Memory adresser
- 5) Funktionskald/maskinkode

Elementerne i enkelte udtryk fortolkes efterhånden som programstumperne udføres. Hver udførelse sker på grundlag af Basic/Debug's oversættelse til et tal.

3.2.2 TALBEHANDLING

Alle regneoperationer udføres i 2 8-bit registre som 16 bit. De 16-bit returneres også som resultat af en regneoperation. Basic/Debug'en fylder op med nul'er i de høje bit hvis kun de lave benyttes. Hvis resultatet af en regneoperation overstiger 16 bit forsvinder det mest betydende bit.

Alle numeriske værdier repræsenteres af 16-bit 2's komplement formen. For negative tal er 2's komplement lig $65536 - N$. Derfor er det mest betydende bit i 2's komplement ON/HIGH for negative tal.

Numeriske værdier går fra minus 32768 til plus 32768. Hvis en regneoperation overskrider minus 32768 bliver tallet positivt - der tælles nærmest i ring.

Hexadecimale værdier benyttes på lige fod med heltal fordi man så nemmere kan finde ud af placeringen i hukommelsen, der som i alle computere, er adresseret i HEX-formen.

Man kan godt adressere helt fra 0 til 65536 i memorycellerne, men man vil kun modtage svar på terminaler i området minus- til plus 32768. I afsnit 3.2.5 viser vi dog et eksempel på udprintning af tal uden for dette område.

For selvfølgelig kan man komme ud over disse begrænsninger, som kun hidrører fra softwareopbygningen.

GOTO G ★ 100

Men GOTO-kommandoen bruges også ofte i direkte kommandoer fordi den faktisk også udfører en RUN kommando. Hvis du skriver GOTO 200 vil programafviklingen begynde fra linie 200.

Men denne form for programafvikling kan være farlig hvis man ikke har tjek på de foregående variabler etc. Programmet kan stoppe og f.eks. skrive:

! ERROR AT LINE 4096 (fejl i linie 4096)

Linie 4096 kan ændres og man kan prøve dens syntax igen ved at skrive GOTO 4096. Husk desuden at GOTO er en hopkommando. Derfor kan den kun benyttes som sidste statement i en linie.

3.3.5 IF/THEN

SYNTAX

if-stmnt => udtryk i relation til udtrykket [THEN]
relationer-op => '<' '>' '|' '<=' '>=' '|=' '<=' '>=' '|='
eftersætning => tal | statement-linie

EKSEMPLER

IF A > B THEN PRINT "A er større end B"

IF A > B "A er større end B"

IF X = Y IF Y = Z PRINT "X = Z"

IF A < B I = 0 : J = K + 2 : GOTO 100

IF I = 2 opstår THEN aldrig - den er falsk

IF/THEN statements bruges hvor man vil sammenligne tal med henblik på et programspring. Sætningen der følger efter kan være et andet statement eller en hel række statements adskilt af kolon'er :

Ethvert statement kan også være en ny IF-kommando og THEN kan evt. udelades hvis du vil spare memory.

IF-kommandoen sammenligner tallet for første udtryk med det andet. Hvis det giver mening kan efterfølgende IF-kommandoer også udføres.

Hvis en relation ikke er sand udføres næste sekventielle instruktion.

Kun i to eksempler kan THEN-kommandoen IKKE udelades. Det gælder hvis den anden instruktion slutter med en decimal- eller hexadecimal konstant, som f.eks. henviser til et linienummer:

IF X < 1 THEN 1000

Ovenstående statement kræver et THEN for at adskille den anden numeriske (1) fra linienummeret (1000). Men THEN kan GODT fjernes hvis udtrykket ser således ud:

IF I > X 1000

Andet eksempel på hvor THEN ikke må udelades er når andet udtryk i et statement er en hexadecimal konstant, og det indgår med en LET-kommando hvor keyword'et udelades for variabler mellem A og F. Et eksempel er:

GO @ kommandoen hører ubetinget til maskinkoderutinerne og må kun bruges når subrutinen ikke sender nogen værdi retur i sig selv. Det første argument er adressen på den første byte i subrutinen. De to efterfølgende benyttes til overførsel af tal til maskinrutinen. I modsætning til USR-funktionen fra 3.2.7.2, bliver indholdet i R18-19 berørt og der kommer intet tal retur. På den anden side overfører GO@ argumentet til subrutinen på samme måde som USR-funktionen efter følgende tabel:

Call	R18-19 indhold	R20-21 indhold
GO @ % 700, A, B	B	A
GO @ % 700, A	A	A

3.3.3 GOSUB KOMMANDO

SYNTAX

gosub—stmnt => GOSUB udtryk

EKSEMPLER

```
GOSUB 50
GOSUB C
GOSUB B x 100
```

I de fleste programmer har man brug for at genbruge småstumper. I stedet for at gentage kommandoerne i en uendelighed sætter man programstumperne sammen i begyndelsen af programmet og hopper til dem efter behov. GOSUB's argument behøver ikke at være et tal, men kan være et udtryk, som peger på linienummeret - f.eks. en variabel. GOSUB må have en returkommando af typen RETURN. Den gemmer nemlig det linienummer som RETURN skal hoppe tilbage til efter at subrutinen er udført.

En subrutine kan kalde en anden. RETURN-instruktionen fra den anden subrutine hopper tilbage til den første. På den måde kan man udføre subrutiner til en dybde bestemt af memory stacken og hukommelsen.

3.3.4 GOTO KOMMANDO

SYNTAX

goto—stmnt => GOTO udtryk

EKSEMPLER

```
GOTO 100
GOTO %FF
BOTO B*100
```

GOTO medfører hop i programafviklingen. Men til forskel fra den gængse Basic vil Z8's Basic/Debug'er acceptere regneudtryk efter keyword. Derfor kan Z8'ens Basic bruge variable som linienummer. Hvis f.eks. variabelen G er 1, 2 eller 3, når linierne 100, 200 eller 300 udføres, vil du kunne bruge instruktionen:

3.2.3 KONSTANTER

En konstant skal repræsenteres af et tal. Konstantens værdi ændres normalt ikke under programafviklingen.

I Basic/Debug kan en konstant tilegnes både decimal- og hex-tal.

Z8'eren benytter de kendte hex-tal: 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E og F. HEX-tallet A er således lig med decimaltallet 10 ligesom HEX-10 er 16 i decimaltal. HEX-tal indskrives som almindelige decimaltal med et %-tegn foran værdien. ALLE andre værdier opfatter den som decimaltal.

Negative tal skal have tegnet » - » foran. Teknisk set opfatter Z8'eren Basic/Debug et negativt tal som » et udtryk » eller en programstump. Minustegnet er en operand og tallet et argument. Det kan være vigtigt at bide mærke i den lille ting i forbindelse med kommandoer uden keyword's. Specielt hvis man udelader PRINT-kommandoen. Se afsnit 3.3.10.

Basic/Debug genkender kun heltal (INT=Integer-tal=heltal). Brøker eller deltal kan ikke indtastes eller modtages retur på grundlag af en kalkulation. Her er et par eksempler på rigtige tal:

```
0      %
123    %7B
256    %100
32766  %7FFE
32768  %8000
```

3.2.4 VARIABLER

En variabel er opbygget som 2 byte i memory'en. Den kendes som et enkelt bogstav, der er dens navn.

Variablen kan ændres når som helst.

Der er 26 variable i Z8. Hvert bogstav i det engelske alfabet er et variabelnavn. Variable ændres KUN når man giver den en ny værdi eller maskinen reset'es. Derfor kan man godt slette et program og køre et nyt med de tidligere variable.

De variable gemmes forskellige steder i hukommelsen som funktion af hvor meget RAM der er til rådighed under opstart. Normalt gemmes de i RAM-TOP fra celle 34 til 85. Da der er to bytes til hver vil variabelen A få pladsen 34-35 og Z 84-85.

Men i minimumsopstillingen med Z8 UDEN RAM, deler variablerne pladsen med GOSUB-stacken. Det kan give problemer i det helt lille system hvis man ikke har styr på RAM-pladsen.

3.2.5 OPERANDER

En operand fortæller Basic/Debug'eren hvad den skal gøre ved de tal der hører sammen med den. Z8'eren operander kan splittes op i 2 kategorier: De aritmetriske og de relative operander:

De Aritmetriske er plus, minus, gange og division.

Regneoperationer med aritmetriske operander foretages fra venstre mod højre. Hvis et udtryk indeholder flere forskellige operander foretages først multiplikation og division, derefter addition og subtraktion. Men det kan dog ændres med paranteser således:

$3 \times 24 - 18 / 3 + 10 = 76$
 eller,
 $3 \times (24 - 18) / (3 + 10) = 1$

Basic/Debug kan ikke klare deltal. Derfor vil operationen i andet eksempel ikke give nogen mente.

Det specielle omvendte » / »-tegn flere terminaler giver kan udføre en division uden fortegn i området 0–65535. F.eks. vil kommandoen PRINT 40000 \ 3 give svaret 133333. Men PRINT 40000 / 3 giver –8512 fordi heltalsværdien af 40000 uden fortegn er –25536.

Det omvendte divisionstegn » \ » behandler dividenden som et 16-bit positivt tal. Derfor vil den medføre et logisk skift til højre i bitmønstrer på følgende måde:

$(-2) / 2 = -1$
 $(-2) \setminus 2 = 32767$

Et forsøg på at dividere et negativt tal med \-tegnet giver et forkert tal.

Tegnet \ (egentlig det engelske Backslash - på danske keyboard: Ø=CHR\$(92)) giver mulighed for at printe højere tal end 32767. Hvis f.eks. N er et tal uden for det lovlige område, kan følgende statements printe større tal:

PRINT N \ ; N-N \ x 10

RELATIVE OPERANDER specificerer forhold mellem størrelser i IF-statements. Der er tale om vurderinger af to eller flere tal i formen: IF A = B THEN..... osv.

Basic/Debug har 6 forskellige relative operander:

= lig med
 <= mindre eller lig med
 < mindre end
 <> forskellig fra
 > større end
 >= større end eller lig med

3.2.6 HUKOMMELSE/MEMORY/ADRESSER

Basic/Debug'en i Z8 kan adressere enhver celle i ind- og udvendig RAM. Derfor kan man både teste og overføre data til samtlige adresser med RAM.

En memoryplads angives ved en signalkarakter (% eller ↑) og en værdi og begge dele kan placeres hvor som helst i et program.

Man peger på enhver memorycelle, ved at sætte det specielle » @ » foran adressen. (CHR\$(64))

Hvis du f.eks. vil pege på den hexadecimale adresse 4096 skriver du @ % 1000. Denne form for byteorienteret kommando benyttes til at skifte om på kontrolregistre i CPU'en, til I/O-kommandoer eller adresseringer.

Du kan også hente 16-bit ord direkte med specialkommandoen » ↑ ». Den henter først den mest betydende 8-bit byte og derefter den næst efterfølgende. En ændring i et pointerregister vil kræve en 16-bit ord instruktion.

Adressens værdi kan være; en variabel, en konstant, et HEX-tal, en AND- eller USR-funktion et udtryk i parentes, eller ved direkte adressering en helt anden memory reference. Et matematisk udtryk udregnes når du har trykket

BASIC/DEBUG UDTRYK & DEFINITIONER

3.3.1 INTRODUKTION til kommandoer

Basic/Debug kan genkende femten keyword kommandoer. De mest benyttede keyword LET og PRINT kan helt udelades hvor argumenterne står alene og faktisk »mangler» dem. Men det finder Z8'eren selv ud af. Hvis du f.eks. vil printe en streng i gåseøjne: "CIRCUIT DESIGN", kan PRINT udelades fordi gåseøjnene fortæller at strengen kun kan have en PRINT kommando foran. Første del af hver af de følgende kommandobeskrivelser definerer den rigtige syntax for dig. Dvs. det du skal skrive til computeren for at den forstår dig. Metasproget brugt til udvikling af operativsystemet definerer reglerne således:

Syntax beskrivelser staves med små bogstaver, hvor det ikke gælder specialtegn. Eksempler herpå er command, stmnt og gosub-stmnt.

Basicsymboler staves med store bogstaver og specialkarakterer i gåseøjne. Eksempler herpå er ', ' LET NEW etc.

Gentagelse af en konstruktion indikeres ved et '+' eller en '*'. Det betyder f.eks. at et tal består af flere sammenhængende cifre.

Paranteser samler en konstruktion som så kan dubleres med et repetitionssymbol som ovenfor - plus eller stjerne.

Firkant klammer om et udtryk betyder at du kan udelade udtrykket efter eget ønske.

Den lodrette streg viser dig at et eller flere udtryk kan følge hinanden.

Anden del af beskrivelsen for hvert keyword beskriver et eller flere mulige eksempler kommandolinier i direkte mode eller programmode.

Efter eksemplerne har du af og til en beskrivelse af specialfeatures.

3.3.2 GO@ KOMMANDO

SYNTAX

go-stmnt	=> GO '@' adresse [', ' arg-1 [', ' arg-2]]
adresse	=> udtryk
arg-1	=> udtryk
arg-1	=> udtryk

EKSEMPLER

GO @ % E000, A, B
 GO @ % 700

3.2.7.2 MASKINKODE FUNKTIONER

Mange vil have brug for at kunne udføre maskinkodeinstruktioner med Z8. Således har vi med denne beskrivelse allerede introduceret næsten 1.000 bytes maskinkode til specialformål. F.eks. keyboardscanning.

Men udvikling i maskinkode er kompliceret og kræver meget stor viden. Derfor skal vi her henvise til ZILOG's bøger om maskinkode på Z8: Z8 Assembly Language Manual og Z8 Assembler User Guide (fås i Medlems-Service). Basic/Debug er i stand til at kalde maskinsprogrutiner med USR-funktionen. Kommandoen GO @, kalder en subroutine uden at returnere en værdi. Mere om det i afsnit 3.3.2.

Hvis du udvikler eller har et maskinkodeprogram til rådighed skal det placeres hvor den ikke skades af GOSUB-staken eller Basic/Debug programmet. Det står der mere om i afsnit 3.6.

Brug adressen på den første instruktion til kaldet af subrutinen i maskinkode:

USR (% 2000)

Basic/Debug'eren udfører alt hvad den støder på fra den adresse i nummerorden. Hvis der ikke ligger maskinkode her kan der ske hvad som helst. Adressen efter USR-kaldet må gerne efterfølges af et eller to tal, som subrutinen skal bruge:

USR (%2000, 256, C)

Adresse og et eller to udtryk skilles med komma'er. Basic/Debug placerer værdierne i register R18-19 og R20-21 (16-bit tal), og forventer at få retur-tal til samme adresser. Den resulterende værdi kan benyttes i et udtryk.

Placeringen af argumenter i registre afhænger af argumenterne. Funktionen USR (%700,A) kalder en subroutine i adresse 700 og sætter register R18-19 til variabelen A. Men skriver du USR (%700,A,B), vil A puttes i R20-21 og B puttes i R18-19. I begge tilfælde må maskinsprogrutinen efterlade sin returværdi i R18-19.

Tabel 3.2.2 USR argumenter og registre

Call	R18-19 indhold	R20-21 indhold
USR (%700, A, B)	B	A
USR (%700, A)	A	A

Maskinkoderutinen må opfylde nogle krav: For det første skal den slutte med en RET-kommando (HEX= AF). Derefter skal den lægge sin retur-adresse i register R18-19, så den kan finde hjem igen, og så kan den bruge ethvert af de frie registre listet i memory-map'en senere i afsnit 3.6.

Register pointeren peger på R16-31 ved indgangen til MC-rutinen, og argumenterne kan hentes fra arbejdsregistre r2-3 og r4-5. Returværdien lægges i r4-r5. Arbejdsregistrenes funktion er en tung omgang og dem må du hente yderligere information om i ZILOG's Microcomputer Technical Manual - eng.

RUN og værdien indgår så som memory adresse eller registertal. Hvis f.eks. den nødvendige adresse afhænger af værdien C, kan Z8 udregne:

145 LET @ (C x 100) = A

Ved indirekte adressering kan Z8 således løbe gennem adskillige adresser og søge den ønskede information. F.eks...

PRINT ↑↑ 8

Den første pil angiver indirekte adressering. Register R-8 er en 16-bit pointer. Den indeholder altid første byte adresse for programmet i memory. For at udføre denne ordre skal Basic/Debug finde adressen i R-8. Den går der til og udprinter indholdet. Pointer registre beskrives yderligere i afsnit 3.6, hvor du også kan se at de kræver hele ord af 16-bit, som i eksemplet nedenfor.

Hvis du ønsker at ændre i processorens registre bruges referenceadresser fra 0-127 og 240-255. Der er ingen registre fra 128 til 239. Derfor gælder også at du kun kan bruge HEX 00 til FF og ertern memory fra HEX 0100 til 00FF. Men brug ikke 16-bit word referencer fra 00FF til FFFF. Det vil returnere helt uvedkommende tal.

Memorykald kan benyttes til oprettelse af data ARRAY's. Man afsætter blot et område i RAM'en og tildeler det ARRAY'et. Næste adresse fås ved hele tiden at lægge een til array'ens startadresse. Hvis vi f.eks. lader en array af byte starte på HEX C000, vil følgende program-statements definere startadresse og de enkelte elementer:

A = % C000

@ (A + J) = 99

@ (A + J) = @ (A + J) + @ (A + K)

Array start adresse

Element J = 99

Hvor A (I) = A (J) + A (K)

3.2.7 FUNKTIONER

Der er to grundfunktioner i Basic/Debug for logiske funktioner: AND, som udfører 8-bit AND-funktion og USR, som kalder maskinkodeprogrammer. Funktionerne kan indgå som operander ligesom variable, konstanter og memory referencer. De ændrer ikke de sædvanlige aritmetriske operationer som plus, minus, multiplikation og division.

3.2.7.1 LOGISKE FUNKTIONER

AND udføres ikke som vi kender det fra normal BASIC. Det er en logisk AND, som det måske kendes fra maskinkode. Den benyttes, hvor man vil slukke eller isolere bestemte bits.

AND (udtryk,udtryk)

De to udtryk i parantesen udregnes hvorefter de AND'es. Man kan for eksempel skrive AND (6,3) hvorefter man får svaret 2.

Hvis man kun skriver et udtryk i parantesen bliver det AND'et med sig selv.

Logisk OR kan også lade sig gøre. Om end ad omveje. Man tager den komplementære bitværdi ved at trække tallet 1 fra. Altså -1.

Hvis vi således vil OR'e A og B sker det på denne måde:

-1-AND (-1-A, -1-B)

